A submission presented in partial fulfilment of the requirements

of the University of Glamorgan/Prifysgol Morgannwg

for the degree of Doctor of Philosophy

A Unified Spatial Data Structure for GIS

Maciej Dakowicz

June 2009

Certificate of Research

This is to certify that, except where specific reference is made, the work described in this thesis is the result of the candidate. Neither this thesis, nor any part of it, has been presented, or is currently submitted, in candidature for any degree at any other University.

Candidate: Matej Dakovic

Director of Studies: C. Gold

Date: 11.05.2010

Abstract

Most GIS systems use separate thematic "layers" to store different types of spatial data. Each of them contains specific characteristics of the area, so there are separate layers for the distribution of buildings, the road network or the relief of the terrain. The spatial information used in GIS can be grouped into four main groups: polygonal maps, terrain models, networks and discrete, unconnected objects. Polygonal maps and terrain models are considered to be "field" models of space, covering the whole map, so there is some kind of information available at every location. On the other hand, networks and discrete objects are representations of the "object" model of space, in which the map is populated by entities and the space between them is empty. Choropleth maps are the most common examples of polygonal maps and the three main representations of terrain models are Triangular Irregular Networks (TINs), grids and contour lines. Networks consist of connected edges, while discrete objects can be points, lines or polygons. In networks, polygonal maps and surfaces there is some model of connectivity available. Polygons are adjacent to each other, as are the elements in terrain models. Network data is connected along the edges and junctions are defined. Unconnected objects need have no connectivity information, but in that case the possible spatial queries are limited.

The layers can be stacked on top of each other to perform various operations and analyses on them. However, there is no consistent method applicable to all data types because GIS has traditionally separated field and object layers and used different data structures to represent them.

This thesis presents a unified spatial data model for these most common types of spatial information and intends to show that it has clear advantages for geographical analysis. The idea is to represent discrete object models as fields, so there is information available at all locations. The model is based on the Voronoi Diagram (VD) and the dual Delaunay Triangulation (DT), two well studied geometric structures. Depending on the application it may be appropriate to represent the data on the map by the simple VD/DT, or their derivatives - the Constrained DT (CDT), the Line Segment VD (LSVD) or the crust and the skeleton. All of these are directly related to each other and may be handled in a single manner in the computer. Algorithms and the storage of these various forms of the VD using the quad-edge data structure is described. This structure may be updated locally, and dynamic algorithms for each of these representations are presented. This allows for the development of a common interactive framework for what are traditionally considered to be distinct data types.

The unified model is illustrated by a variety of GIS applications, and the implementation of several traditional GIS operations and queries is discussed.

Background of This Work

My collaboration with Prof. Christopher Gold dates back to the beginnings of the year 2000. I was still a student at Technical University in Bialystok, Poland trying to complete my Master degree in Computer Science. The subject of my dissertation was "Generation of Terrain Models from Contour Lines" and I was looking on the Internet for papers on that subject and people working on similar issues. One of the persons I contacted was Prof. Christopher Gold, working at Laval University in Quebec in Canada. We started exchanging emails in which he gave me some very useful hints, helping me to finish my thesis. After I was awarded my Master's degree Chris asked whether I would like to work for him on Voronoi Diagrams in Hong Kong, since he had just moved there from Canada and was looking for a Research Assistant, who understood Voronoi diagrams and terrain modelling. After the first shock and some days of hesitation I eventually said "Yes, why not?" and on the 31st of October I met Prof. Chris Gold in person in Hong Kong. I worked for him at The Hong Kong Polytechnic University until the middle of 2004 and then followed him to Wales to continue working on Voronoi Diagrams and start my PhD.

The work presented here is the result of eight years of close collaboration. Due to the scope and size of the project it is very hard to completely separate our contributions. The overall concept, that a unified model of many GIS operations could be developed based on the Voronoi diagram, had been suggested by Prof. Gold some years ago, but the amount of work required to turn it into reality prevented a full realization of the project prior to our collaboration. My original contribution has been the detailed analysis of these general concepts and turning them into reality. This consists of the development of all algorithms, including the extremely difficult Line Segment VD, and production of the software to demonstrate their validity. This required spending many hours discussing the details of various algorithms and solving all related problems. A significant part of this thesis (Chapter 5) describes the practical development of various Voronoi algorithms and demonstrates the complexity of the project, while Chapter 6 describes the value of Voronoi diagrams as a consistent spatial model for many of the GIS operations in a wide variety of applications.

Apart from the thesis, the work produced consists of thousands of lines of code and it demonstrates that both object and field models can be formed, managed and analysed using one type of structure and a single set of algorithms. A consistent spatial model also allows interaction between different GIS types, which opens up many new applications and possibilities. I hope that the set of tools described in this work will motivate a re-evaluation of the spatial models currently used.

About this Thesis

The work described in this thesis represents the results of research I carried out at the GIS Research Centre, Faculty of Advanced Technology, University of Glamorgan, Wales, UK from October 2004.

The kinetic VD/DT/CDT/LSVD algorithm (described in Chapter 5) and its applications were published in four papers:

- Gold CM and Dakowicz M (2006). Kinetic Voronoi/Delaunay drawing tools. In *Proceedings of the 3rd International Symposium on Voronoi Diagrams in Science and Engineering (ISVD06)*, pages 76-84. Banff, Canada.
- Dakowicz M and Gold CM (2006). Structuring kinetic maps. In Reidl A, Kainz W, and Elmes G, editors, *Progress in Spatial Data Handling - The 12th International Symposium on Spatial Data Handling*, pages 477-493. Springer-Verlag Berlin.
- Gold CM and Dakowicz M (2007). Dynamic cartography using Voronoi/Delaunay methods. In *Proceedings of the 5th ISPRS Workshop on Dynamic and Multi-dimensional GIS (DMGIS07)*, pages 41-47. Urumqi, China.
- Gold CM, Mioc D, Anton F, Sharma O, and Dakowicz M (2008). A methodology for automated cartographic data input, drawing and editing using kinetic Delaunay/Voronoi diagrams. In Gavrilova M, editor, *Generalized Voronoi Diagram: A Geometry-Based Approach to Computational Intelligence*, pages 159-196.

The interactive terrain modification method (described in Section 6.5.4) was published in two conference papers:

- Dakowicz M and Gold CM (2005a). Concepts of interactive terrain modification. In *Proceedings of the 13th Annual Conference of GIS Research UK*, pages 252-257. Glasgow, Scotland.
- Dakowicz M and Gold CM (2005b). Interactive TIN modification with a cutting tool. In *Proceedings of the 4th ISPRS Workshop on Dynamic and Multi-dimensional GIS (DMGIS05)*, pages 5-9. Pontypridd, Wales, UK.

The Voronoi based runoff modelling (Section 6.5.5) was described in two conference papers:

- Dakowicz M and Gold CM (2007a). Finite difference method runoff modelling using Voronoi cells. In *Proceedings of the 5th ISPRS Workshop on Dynamic and Multi-dimensional GIS (DMGIS07)*. pages 55-60, Urumqi, China.
- Dakowicz M and Gold CM (2007b). Finite difference runoff modelling using "Voronoi buckets". In *CISIM '07: Proceedings of the 6th International Conference on Computer Information Systems and Industrial Management Applications*, pages 121-124. Elk, Poland.

Some applications listed in Chapter 6 were described in:

- Gold CM and Dakowicz M (2005). The crust and skeleton - applications in GIS. In *Proceedings of the 2nd International Symposium on Voronoi Diagrams in Science and Engineering*, pages 33-42. Seoul, Korea.

The Marine GIS application of the kinetic Voronoi diagram (Section 6.2.5) was presented in:

- Goralski R, Gold CM, and Dakowicz M (2007). Application of the Kinetic Voronoi Diagram to the real-time navigation of marine vessels. In *CISIM '07: Proceedings of the 6th International Conference on Computer Information Systems and Industrial Management Applications*, pages 129-134. Elk, Poland.

Acknowledgments

Firstly, I would like to thank Prof. Chris Gold, and his wife Valerie for changing my life completely and opening the world to me by offering me a research job in Hong Kong, and later the PhD scholarship in Wales. His passion for Voronoi diagrams has been so strong and so contagious that we have been working together for eight years already. My deepest thanks for all I have received from him, I simply could not have a better supervisor than Chris. It is impossible to list everything I would like to thank him for here, all his support, patience, friendship, encouraging me to finish writing up the thesis and correcting it with Valerie. However, one thing I would like to thank him for especially is his tolerance for my passion of photography and allowing me to travel for weeks to all exotic places. Thank you Chris.

I would also like to thank my old friend Dr Hugo Ledoux, who is another Voronoi enthusiast, with whom I discussed many Voronoi problems and issues and with whom I share many memories from Hong Kong, Wales and other places.

A big thank you to Dr Rebecca Tse who took care of me like an older sister (even though she is younger than me) when I arrived in Hong Kong for the first time, and who was always helpful when needed.

My thanks also goes to Pawel Boguslawski who motivated me to work harder during these years of PhD study in Wales, and who shared his car with me on the way from Cardiff to the university, even though my place was not on the way.

I would also like to thank my parents for their love, for giving me a good education in Poland and letting me leave them to go and work for Prof. Gold.

Finally, I would also like to gratefully acknowledge The University of Glamorgan for the financial support of this research.

Contents

List of Figures

Chapter 1

Introduction

1.1 Representing Geographical Data

Let us imagine a fragment of land with a rather large village in the middle. To describe it to another person we would speak about different spatial features present there. We would say that there is a river flowing across the whole area, branching into smaller rivers. There are some roads and various buildings - houses, a church, a post office and a couple of shops. There are lamp posts on the side of some roads and a couple of mailboxes. The village lies in a small valley and is surrounded by forests and arable fields and there are also some farms nearby. It is summer time and there are various types of crops being grown in the fields.

We could also describe the area by drawing it on paper. We would draw points to mark the positions of lamp posts and mailboxes. We would draw lines to mark the river and road networks, and rectangles and more complex polygons to draw various buildings. Then the whole area could be divided into physical or virtual blocks according to the postcode information, land ownership, or to mark the boundary of the village and different farms around it. The land of farmers could be also divided into various blocks, depending on the types of crops being grown. However, depicting the relief of the underlying surface using symbols would definitely be a tricky task. Additionally, depending on the scale, the map features can be represented differently, for example rivers can be drawn using lines of the same width, but at a large scale they can be represented as lines having various width at different locations along the river, or even areas. Nevertheless, by drawing all those features we would end up with a map of the area. Having even such a simple map we can use it to identify its features, roughly estimate distances between houses, count the number of lamp posts or identify neighbouring farms.

When representing such an area in a computer GIS (Geographical Information System) the majority of the real world phenomena can be grouped in four main categories, as shown in Figure 1.1. The simplest are points used to represent objects having relatively small size, too small to be represented as polygons, such as the previously mentioned lamp posts or mailboxes. Series of pairs of points make lines, that can be used to represent roads or geologic faults. Another structure is a polygon, that can store relatively large single objects, usually with clearly defined boundaries, like houses or lakes. Points, lines and polygons can be grouped as discrete objects. Additionally, some of the objects can be mobile, such as people or cars. Another category are polygons (areas) covering the whole area of interest, storing parts of land adjacent to each other, like postcode areas or properties. The third category are networks, which are a special case of lines, having defined

topologies. Networks consist of connected lines and are used to model flow in rivers or roads. The last category is surfaces, that store the information about the relief of the terrain or similar continuous attributes, such as the temperature.

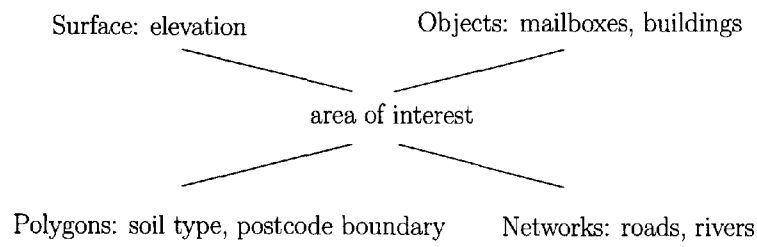


Figure 1.1: The traditional spatial entity model (after Heywood et al. (2002)).

Some structures cover the whole area (like space partitioning polygonal maps), while others are only present at specified locations (points, polygons of buildings or networks). Some also provide information about relationships between represented objects (topological structures), and others not. A point itself contains only the information about its location and some attributes, it does not "know" whether there are other points located in its vicinity. A similar situation is the case of polygons, for example representing single houses. Each polygon has clearly defined boundary providing information about the location of each house, but there is no explicit knowledge about other houses. In both representations, there is no information available about locations other than the points or polygons represented. When asking a "what is at this location?" question we receive an answer if there is an object present there, otherwise the answer is "nothing". A slightly different situation is the case of networks, which consist of lines connected at common nodes. There is information about connectivity in the network available and various analyses can be performed, but there is nothing known about the area in which the network itself is located. On the other hand, in the model of space partitioning polygons, the whole map is covered with blocks, adjacent to each other, sharing nodes and edges, so there is connectivity information (topology) available. Each polygon is associated with some attributes and the information about neighbouring polygons is available. Similarly, in the case of surfaces, the whole area is covered with polygons, being usually triangles or squares. It is possible to obtain the elevation value at any location. There is also some neighbourhood information available for adjacent cells, depending on the surface representation model.

The discussion of various data structures and the way they cover the area leads us to the two different and widely recognized ways of representing the geographical world (Goodchild (1992)). One is called the discrete object model, where the world is populated with discrete objects. Those include houses or mountain locations and are modelled in GIS using simple features, such as points, lines and polygons. Objects are not related to each other and are identified by their attributes. Any point in the plane can be a part of zero, one or more objects, so objects can overlap and they do not need to cover the whole space (Goodchild (2001)). Another is the field model, with the map covered completely by a finite set of variables. Surfaces and polygonal maps are representations of fields and the real world examples include maps of soil type, land ownership or elevation. At each location there is an attribute value available so the question "What is here?" always produces an answer. Field models can be classified as having discrete attributes (the same value for a specified

area, like land ownership or vegetation cover type) or continuous attributes (varying smoothly, like temperature or elevation). To be able to model fields in computers, they have to be discretised (broken into finite parts), since they are continuous and computers are discrete machines. A field can be partitioned, or tessellated, into regular or irregular cells, so the whole area can be covered with triangles, squares or other polygons. There is always some kind of adjacency available, since objects share boundaries, and the adjacency relationships if needed can be stored in the objects or calculated on-the-fly.

Most GIS systems use separate thematic "layers" (overlays) to store different types of data, which can be of object or field type. Each layer contains specific features or characteristics of the area, so there is a separate layer for the road network, the distribution of buildings or the terrain relief. The layers can be stacked on top of each other and various operations can be performed on them. Some of the simplest are queries using a single layer, including finding what is present at a specified location, what is the elevation at a specified point or what is the area of a selected parcel. Using more than one layer the operations can be performed on different objects and characteristics, so we can identify the nearest mailbox to the selected house, find houses located within a specified postcode area or mark areas of the city with the highest population density.

However, although it is possible to compare and perform operations on different layers, there is no consistent method applicable to all data types because GIS has traditionally separated field and object layers and used different data structures to manage them (Burrough and McDonnell (1998)). Additionally not all operations can be performed on both types of layers. In field models there is always adjacency information available, as cells share common boundaries. Such information is not present for discrete objects. This creates problems for queries which can only report presence or absence of an object at a specified location, and cannot locate adjoining objects. Thus GIS operations requiring proximity and connectivity information are more difficult with object models and usually require additional spatial indexing structures. These include interpolation, where the attribute for the selected location is estimated based on the attributes of a specified set of neighbours. Also simulating the movement of point objects is problematic, as it requires knowledge about neighbouring objects in order to avoid collisions.

Theobald (2001) states "that the data structure used to represent spatial features is important in a GIS because it determines the range of functions and analyses that are easy to provide". Using a unified data structure able to represent all spatial data would make most of the GIS operations possible for any type of the input information.

1.2 Objectives of this Work - The Unified Spatial Model

The main objective of this research is to develop a unified spatial model for objects and fields and demonstrate its usefulness with various traditional GIS operations and analyses. The solution is to represent both models using proximal maps. This would standardize the GIS operations and make the topology information available in every model. Each object in a proximal map is associated with a region enclosing the part of the map closer to that object than to any other. The adjacency between objects is clearly defined, as neighbouring objects share boundaries. Converting discrete object layers into proximal maps transforms the query "What is here?" to "What is closest to here?", so the answer is available at any location.

The basis of this this work is a vector Voronoi Diagram (VD), which is a proximal map of the input data (solid lines in Figure 1.2). The VD is considered to be "the fundamental spatial data

structure" (Aurenhammer (1991)) and the creation of the VD of a set of discrete points converts the model to a continuous field. The resulting diagram covers the whole map, the attribute value at any location inside each cell is available from its node and the adjacency relationships between nodes are clearly defined via shared edges. The VD is associated with its dual graph having edges connecting neighbouring points - the Delaunay Triangulation (DT, dashed lines in Figure 1.2)). Both are well studied and there are many algorithms allowing their construction and modification (Guibas and Stolfi (1985); Devillers (1998)). It is possible to extract from the VD/DT two other geometric structures being subsets of the VD/DT edges. These are the crust and skeleton (Amenta et al. (1997); Gold and Snoeyink (2001)) and have many interesting properties. They can be used in various GIS operations, including digital terrain construction or watershed generation. Additionally, it is possible to convert the VD/DT models to rasters using various interpolation techniques to perform traditional analysis on them, such as slope estimation.

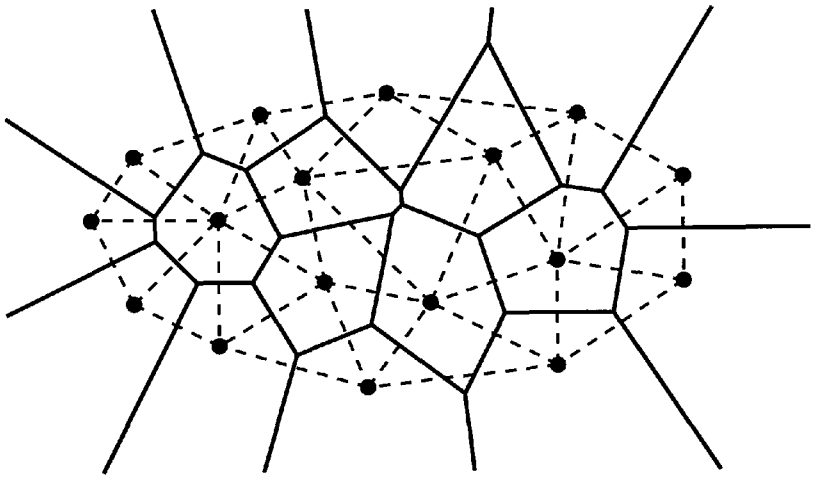


Figure 1.2: Voronoi diagram (dashed lines) and Delaunay triangulation (solid lines) of a set of points.

However, modelling line and polygon objects present in polygonal or network maps with the ordinary point Voronoi diagram is problematic, as the connectivity of two nodes in the VD depends on the distance between them and the configuration of neighbouring nodes. The ordinary VD does not guarantee preserving connectivity between selected nodes and this is why the Constrained Delaunay Triangulation (CDT) (Lee and Lin (1986); Chew (1987)) and the Line Segment Voronoi Diagram (LSVD) (Gold et al. (1995); Imai (1996); Held (2001); Karavelas (2004)) were developed. Both structures preserve the configuration of the input line segments, but in different ways. In the CDT the input segments (constraints) are "forced" into the triangulation as edges (Figure 1.3a) and the only thing distinguishing constrained edges from ordinary edges is a flag, stating whether the edge is constrained or not. The resulting triangulation is not fully Delaunay and the structure of its dual Voronoi diagram is different from the ordinary VD. By contrast, in the LSVD (Figure 1.3b), the input segments are separate objects and each of them has an associated Voronoi region, just like point objects. However, the Voronoi boundary between a point and a line segment is a parabola, so the edges of Voronoi regions can be parabolas, as well as straight lines.

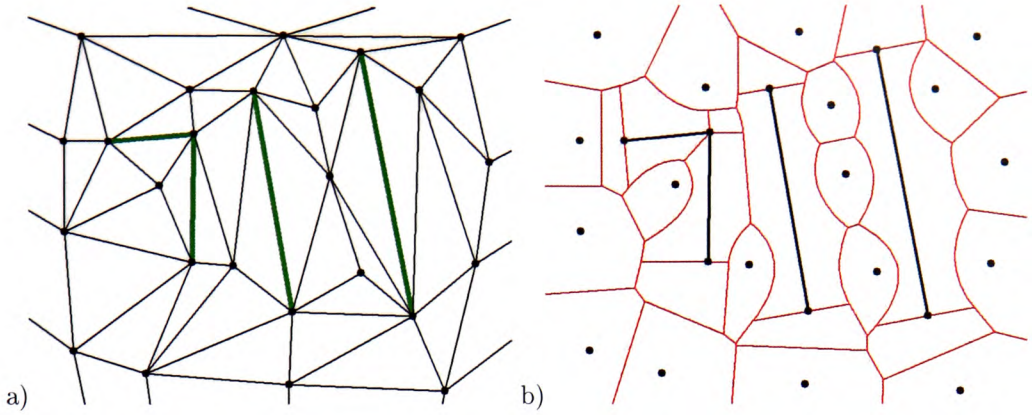


Figure 1.3: The CDT (a) and LSVD (b) of the same set of data with four input segments.

This work presents dynamic methods that can be used to construct and update the ordinary VD/DT as well as the more complex CDT and LSVD diagrams. They are based on the idea of moving points in the VD (Gold (1990); Roos (1993); Gavrilova and Rokne (1999)). In the case of the CDT/LSVD the moving point leaves a trace behind, which becomes the new line segment (Gold et al. (1995)) or constrained edge. Depending on the applications, the input data and intended operations, the ordinary VD/DT, CDT or LSVD can be produced from the input set of points or segments and points. All these types of Voronoi diagrams are constructed using the same idea and mechanism. Important in practice local updates are possible at any time using operations of insertion and deletion of points and line segments, whereas the known methods of Held (2001) and Karavelas (2004) allow only insertion of line segments, without deletion.

Various types of spatial data, as shown in Figure 1.4, can be used in GIS to form proximal maps and be stored and manipulated in the same manner. Discrete points are converted to fields by calculating the ordinary Voronoi diagram, while lines and discrete polygons by obtaining the CDT or LSVD. Networks are used to construct the LSVD or CDT using their segments. These diagrams preserve connectivity of the links (so flow can easily be simulated) and add the possibility of additional analyses, such as the proximity to the nearest network segment. Polygonal maps are field models already, but representing them with the LSVD adds topology (connectivity relationships) to the map. Surfaces, which are also field models, can be constructed from points, contours or rasters. Representing them with the VD/DT model is a common GIS solution and has many well documented advantages.

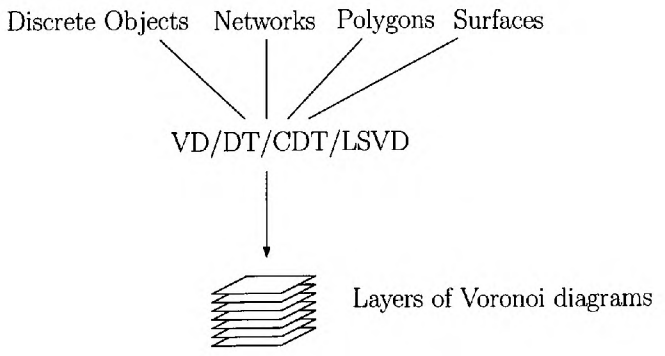


Figure 1.4: Unification of various models in GIS using Voronoi diagrams.

All these Voronoi based maps can be used to perform the traditional GIS operations and analyses (Burrough and McDonnell (1998); Longley et al. (2001)). Various examples are provided for each type of the input data. Finally, operations requiring more than one map, such as queries and merging operations (the traditional overlay, Bolstad (2003)), can be performed on various combinations of Voronoi layers. "Query" extracts information from one map using data from another, while "merge" combines two layers. Merging (the overlay) using the VD is performed by drawing the secondary layer onto the primary layer, snapping lines or points together whenever collisions, or close collisions, occur. All these operations can be performed in a consistent and straightforward fashion as the structure of all input layers represented by different Voronoi diagrams is the same.

We would like to point out that spatial data could be divided into more than four categories. However, the four used in the thesis include most types of the data. We are also not focusing on mobile objects, although point relocation mechanisms are provided. We are not concerned about time. Also the focus of this work is on the topology of the mesh, not on the attributes of objects. We operate on the 2D data (or 2.5D) and provide definitions only for 2 dimensional cases.

1.3 Outline of the Thesis

This thesis consists of seven chapters, including this one. It can be split into five main parts: an introduction to the problem (Chapter 1), the previous work in this field (Chapters 2, 3 and 4), the new work (Chapter 5), applications (Chapter 6) and conclusions (Chapter 7).

After the description of the work motivation and thesis objectives in Chapter 1, Chapter 2 presents concepts needed to understand the task. Space and its two different conceptualization models of fields and objects are explained, together with their different raster and vector representations. Additionally the term "GIS" is introduced, together with spatial analysis and its different taxonomies and operations.

Chapter 3 focuses on computational geometry and introduces the structures used in further work. The explained concepts include those based on points the Voronoi Diagram and the Delaunay Triangulation, together with related crust and skeleton structures. The idea of points moving in the VD (kinetic VD) is briefly explained. Additionally, concepts of the Constrained DT and the Line Segment VD, allowing incorporation of line segment features into the diagram, are introduced.

Chapter 4 focuses on technical aspects of the construction and management of the diagrams presented in Chapter 3. Various data structures used to store and manipulate Voronoi diagrams and triangulations are presented and the quad-edge data structure used in the project is explained in detail. The existing algorithms for constructing and updating the VD/DT diagrams are reviewed. They include the management of moving points, the insertion and the deletion of points, constrained edges and line segments, as well as the object location and mesh traversal techniques. These operations are the basis of the unified VD/DT data structure for GIS, developed in Chapter 5.

Chapter 5 explains in detail the new kinetic method of DT/VD/CDT/LSVD construction and modification. One unified methodology is used to build all the diagrams - the VD/DT, the CDT and the LSVD. The algorithms are divided into four groups according to the level of their complexity. The first group is basic tools, and its methods include simple arithmetic operations such as distance calculations or orientation tests. There are also some complex operations, including an important calculation of the circumcircle of three objects, as well as mesh traversal and point

location operations, which all have been specifically developed in this research to work on all types of the VD/DT. The second group is the low-level operators. They include functions used to modify the mesh directly such as "Split" to split a new node from an existing one, "Merge" to join two nodes into one and "DisconnectLineSegment" to detach a line segment from its endpoint. They also include edge to line segment and line segment to edge conversion operations ("ConvertEdgeToLineSegment" and "ConvertLineSegmentToEdge"). The third group consists of only one method - the procedure "MovePoint", which is responsible for relocating a node from one location to another: the trace that the moving point leaves behind can be used to insert or remove constrained edges and line segments. The fourth group is the high-level operators available to the user, that are used to construct and modify the mesh. They include relocating nodes, and inserting/deleting points, constrained edges and line segments. These are basically calls to the procedure MovePoint with different parameters. Features are inserted using the split and move point mechanisms, and deleted by moving and merging points. The chapter ends up with the discussion of the robustness issues and examples of results of the algorithms.

Chapter 6 presents practical applications of the unified data structure in GIS. It starts with a discussion about the CDT and the LSVD based spatial models which shows why the LSVD is more appropriate for representing sets of objects built from points and lines than is the CDT. Then examples of the four main types of spatial information represented by various types of Voronoi diagrams are provided. Then the traditional GIS operations and applications are described for each of the spatial data types (such as reclassification, network analysis or terrain modelling), together with additional operations that are possible thanks to the employment of the VD/DT structure. The final part of the chapter describes operations possible on a group of layers representing different kinds of spatial information. Two operations are explained - "query" (the traditional "point in polygon" operation) and "merge" (the traditional GIS "polygon overlay"). These operations, in contrast to the traditional GIS requiring various data structures and methods, can be performed in a consistent and straightforward fashion using layers represented by Voronoi diagrams.

Chapter 7 concludes the thesis by summarizing the work undertaken, presents advantages of the unified spatial model and discusses limitations and future works.

The Appendix consists of two parts and contains a detailed description of the iterative circumcircle calculation method and the moving point algorithm, briefly described in Chapter 5.

There is some code and pseudo code in the thesis written using Object Pascal notion. I believe it helps to understand some technical details of the implemented processes.

Chapter 2

GIS Background

This chapter presents some background information about GIS and concepts associated with it, that are related to the topic of this research. It starts with several different definitions of GIS and its history. Then the concept of spatial data is introduced, together with four major types of spatial data categories that all are later modelled with Voronoi diagrams in Section 6.1. These are discrete objects, networks, space exhausting polygons and surfaces. They are representations of field or object conceptual models, depending whether they cover the whole plane (fields) or empty space exists between them (objects). Field models in order to be managed in computers are converted to tessellations, while object models to sets of discrete entities. They can be be modelled using raster or vector representations. The term of spatial analysis is also introduced, together with various classifications. The chapter concludes by highlighting problems associated with the current GIS architecture that motivate this work on a unified data structure.

2.1 GIS History and Definition

The GIS acronym stands for Geographic Information System. It is a system that allows integration, storage, manipulation, analysis and display of geographically referenced information. Basically, we can say it is a computer system to simplify working with maps.

The modern GIS dates back to 1962 when Dr. Roger Tomlinson developed the "Canada Geographic Information System" (CGIS). The system consisted of a mainframe computer and associated peripherals. It was used by the federal Department of Forestry and Rural Development to store, analyze, and manipulate information collected for the Canada Land Inventory (CLI). The land capability for rural Canada was determined by processing input information about soils, agriculture, forestry, and land use. The system using a national coordinate system produced layers for different themes and provided operations for the overlay of polygons, measurement of areas, and also for digitizing and scanning of maps. It also stored the attribute and location data in separate files. The system lasted into the 1990s and many of the ideas and features introduced by CGIS are still used in modern GISs.

In the early 1980s companies like Bentley Systems, Intergraph or Environmental Systems Research Institute (ESRI) started developing their own commercial GISs, often based on ideas introduced in CGIS and enhanced using database solutions. The first version of ARC/INFO released by ESRI in 1982 for minicomputers is often considered as the first modern GIS. ESRI's ArcInfo (later ArcGIS) and Bentley's MicroStation became very successful products. The latest trend is

development of free, open source GIS packages which run on various platforms. Also the Internet is becoming a new platform for GIS, as there is a growing number of web based geographical applications and it is also one of the main sources of data.

There are many formal definitions of GIS. Burrough and McDonnell (1998) list as many as eight of these including the one saying that "GIS is a powerful set of tools for collecting, storing, retrieving at will, transforming and displaying spatial data from the real world for a particular set of purposes". It can also be defined as "a computer-based system that integrates the data input, data storage and management, data manipulation and analysis, and data output for both spatial and attribute data to support decision-making activities" (Malczewski (1999)).

The functionality of GIS can be divided into four groups: data input - the process of identifying and collecting the data; data storage and management - functions needed to store and extract data from the system; data manipulation and analysis - operations on data; and data output - display of the results of GIS operations (Malczewski (1999)).

2.2 GIS Data

Geographical (or spatial) data is the raw material that can be associated with a location on the Earth's surface and include facts, results of observations, remote-sensing images, census figures and statistics. Data in order to be considered useful needs to be transformed into information. Data become information when it is "organized, presented, analysed, interpreted and considered useful for the decision problem" (Malczewski (1999)). Geographical information can be defined as georeferenced data that has been processed into a form meaningful to the decision maker and is useful in the decision making process.

Today the majority of spatial data is stored in digital format and GIS seems to be a natural choice for its visualizing and processing. Most traditional maps have been converted to digital format by scanning and digitizing them. Digital maps allow viewing at various scales and seamless transition to adjoining areas using zooming and panning techniques. Animation and 3D display are possible making them more attractive and more interactive, allowing better visualization of map features or terrain relief. They can also be easily extended incorporating additional data. Digital maps can also be integrated with a database and be grouped into layers. This allows efficient analysis and better decision making.

2.2.1 Sources of Data

Data collection is one of the most important GIS tasks. There are many sources of data and many ways of entering them into a GIS database. Geographic data capture methods can be grouped as primary and secondary, for both raster and vector types (Longley et al. (2001)). Primary data is measured directly in the environment and created specifically for use in GIS. Those include digital remote-sensing images and aerial photographs for rasters, and GPS and survey measurements for vectors. Secondary data was created for another purpose and has to be processed in order to be usable in GIS. These include scanned maps, photographs or digital elevation models from maps for rasters and topographic maps and toponomy databases for vectors.

Another method of obtaining data is transfer from external sources. There are many commercial and non-commercial data providers. Now the Internet is the main source of data and most geographic agencies and organization have on-line geographic data catalogues. There are

also Internet stores specialized in providing geographic data.

2.2.2 Spatial Data Categories

In most GIS there are four major categories of features and related data structures:

1. Discrete objects. They can be points, lines or polygons (Longley et al. (2001)). Points are used to represent locations of features or objects having relatively small size, too small to be represented as polygons. They can indicate spatial occurrences of events (fire occurrences) or locations of objects, such as lamp posts or mailboxes. Lines can be curved, but these are usually are represented in GIS as series of straight-line segments connecting points. They are used to represent linear entities, such as roads or geologic faults. Polygons store relatively large single objects, usually with clearly defined boundaries, such as buildings or lakes. Some objects can be mobile, changing their position in the map, such as people or cars.

2. Networks. They are a special case of connected lines with defined topologies. Networks consist of segments and nodes at the locations where segments join. They are used for example to model flow in rivers or roads.

3. Polygons (planar partitions). They are space exhausting blocks, so the the whole area is covered by non-overlapping polygons. Each polygon consists of a closed loop of straight lines or curves (approximated usually by series of straight lines). Examples include postcode boundaries or land ownership parcels.

4. Surfaces. They store information about the relief of the terrain or other field information, such as temperature.

All these four categories of features are usually stored in GIS using different data structures and manipulated using different techniques which significantly increases the complexity of the system. For example calculating the distance between two locations is performed differently for networks and surfaces, as they are stored using different structures.

2.3 Space

The term "space" is not easy to define. Its nature has been debated for a long time and there is not a definition accepted by everybody. Intuitively it is the space in which our bodies move and where everything is located. Worboys and Duckham (2004) provide a summary of important overviews of the term. A distinction is often made between small-scale space that can be apprehended by humans using visual perception and large-scale that cannot be observed at once. They list the classification of the term by Zubin (1989) into four types according to the size of objects populating the space with reference to human perception. His A-space contains objects manipulable in everyday life and can be viewed from one perspective, such as pens or books; B-space contains objects larger than objects in A-space that cannot be manipulated by humans and cannot be perceived from one perspective, such as cars or houses; C-spaces are landscapes that can be perceived from one viewing point, such as a scenic overlook; D-spaces are too large to be directly perceived and locomotion is required to experience them. Similarly Freundschuh and Egenhofer (1997) synthesized six topologies of space using the properties of manipulability, locomotion, and size of space:

manipulable object space, non-manipulable object space, environmental space, geographic space, panoramic space and map space.

In the context of GIS the space often refers to geographical space, which is "the structure and properties of the relationships between locations at the Earth's surface" (Worboys and Duckham (2004)). Examples include cities or countries (which are too large to be observed or manipulated by people).

To be able to determine locations and define relationships between objects, a coordinate system is necessary. The most common coordinate model of space is Euclidean space. It is a metric model, so distances between points and angles between vectors can be defined. In two dimensions Euclidean space is known as the Cartesian plane, denoted $\mathbb{R}^2$. A coordinate system can be defined by a pair of perpendicular axes intersecting at a fixed origin.

2.4 Geographical Phenomena in Space

Geographical phenomena can be described in the real world using "what" and "where" information - what is present and where it is located (Burrough and McDonnell (1998)). de By (2001) extends it with "when" information, defining a phenomenon as something that can be named or described, can be georeferenced and can be assigned a time at which it is/was present. If the time information is missing when describing a phenomenon it means that the phenomenon is always present. Common geographical features include rivers, towns, elevation or soil type. Some of them can be explicitly defined and exactly located, like buildings which have well defined boundaries. Others, like hills, do not have an exact form and their boundaries often can not be specified, which is even more apparent when modelling soil variation or temperature. This distinction can be used to classify the phenomenon as being either of "object" or "field" type. In the object model of space it is populated by separate entities, while the field model represents space as a continuous domain of attributes (Goodchild (1992), Worboys and Duckham (2004), Longley et al. (2001)). Entities in the object model are well distinguishable bounded objects described by their attributes, located using a geometric coordinate system and empty space can exist between them, while in the field model the represented phenomenon is present at all locations of the space. Objects are often man-made phenomena, while fields are often natural geographic phenomena, although many exceptions exist.

2.4.1 Field Models and Tessellations

A field is geographic phenomena present at every location in the study space (de By (2001)). The usual examples or fields are temperature and elevation, which are continuous geographic phenomena. For soil classifications or postcode areas there are also values available at every locations, but they are constant within bounded regions. Thus fields can be classified as discrete or continuous (de By (2001)). Discrete fields divide the space into mutually exclusive regions, each having the same field value inside, so there is a steep change in the field value when moving from one region to another. On the other hand, in continuous fields changes in the field value are gradual, as the underlying function is assumed to be continuous. A field-based model is collection of fields.

Fields in order to be represented in the computer, which is a discrete machine, have to be converted into tessellations. These are subdivisions of space into a set of disjoint objects, constructed from points and lines. In 2D these objects are polygons. Tessellations are usually created from

samples of the fields by arranging them into a coherent structure with specified rules to obtain values at all locations.

Worboys and Duckham (2004) provide a simple classification of field models, dividing them into two categories - regular and irregular tessellations. Ledoux (2006) adds "hierarchical tessellations" category to this classification. Regular tessellations are built of polygons having the same shape, while the shape in irregular ones may vary. Grids are the most popular regular tessellations and TINs are the most popular irregular.

Goodchild (1993) provides a different, more detailed classification (as in Figure 2.1), stating that fields are usually modelled in one of the six ways:

1. Irregular point sampling - the database contains coordinates of irregularly spaced sample points representing the variable, e.g., weather station data.

2. Regular point sampling - as in 1 but samples are regularly spaced, e.g., digital elevation model.

3. Contours - the database contains chains of connected nodes with a certain attribute, e.g., digitized contours of elevation.

4. Polygons - the area is divided into a set of space exhausting, non-overlapping polygons with a constant attribute, e.g., soil map.

5. Cell grid - the area is divided into regular grid cells with a value attached to each cell, e.g., a remote sensing image.

6. Triangular net - the area is divided into irregular triangles with the values of the variable specified at each of the vertices and varies linearly inside each triangle, e.g., the Triangulated Irregular Network (TIN).

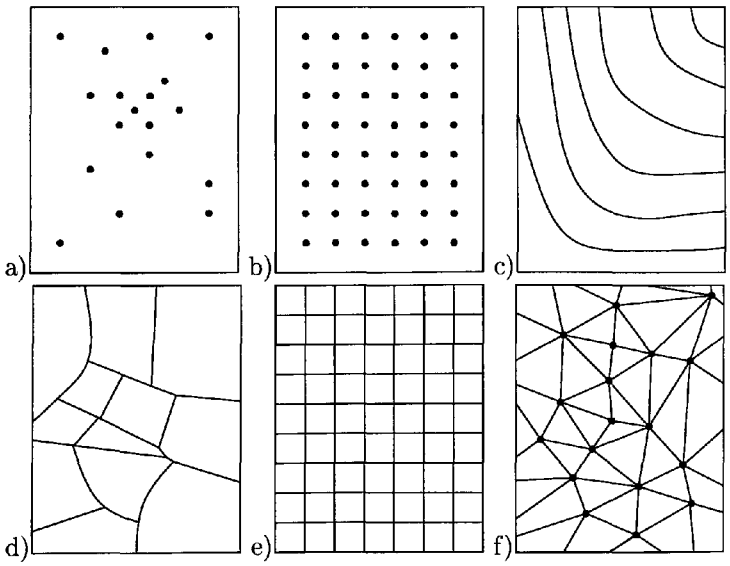


Figure 2.1: Six traditional representations of field data. a) Irregularly spaced points. b) Regularly spaced points. c) Polylines representing contour lines. d) Irregular polygons. e) Regular cells. f) A TIN network.

Cases 1-3 of discretisation are termed incomplete representations of fields, and cases 4-6 complete. This is because in the first three types for query points not being sample points nor a part of a isoline some form of interpolation is needed to obtain an estimate. In the remaining three cases an estimate is available at any location.

However, these six categories are not mutually exclusive. The cell grid models are a special case of polygon subdivisions, so they are conceptually the same, and the only difference between two point sampling models is the pattern of samples. Also the values between sample points are not clearly defined. These are the reasons, why the tessellation based classification of Worboys and Duckham (2004) will be used in this thesis. Although it is simpler and more general, it seems to cover all possible field representations without redundancy.

Tessellations are usually divided into regular and irregular, depending on the shape of their cells. A regular tessellation is a partition of the plane into non-overlapping regular polygons of the same size (Worboys and Duckham (2004)). Regular polygons have edges of the same length and all internal angles are equal. The shape of cells can vary from regular triangles or squares to hexagons and more complex polygons, as in Figure 2.2. Grids, discussed later in this chapter, are the most popular regular tessellation.

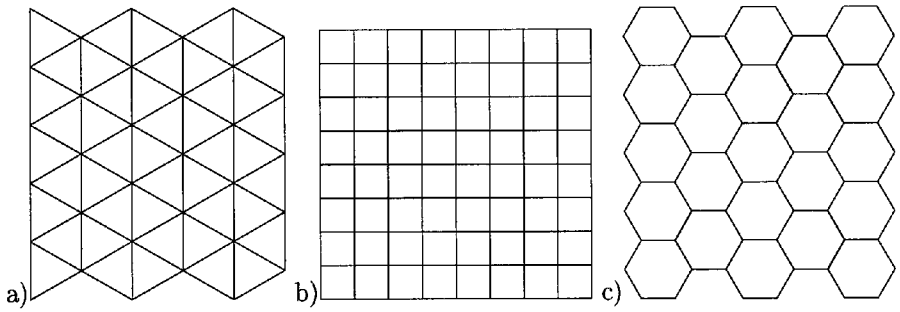


Figure 2.2: Regular tessellations. a) Equilateral triangles. b) Squares. c) Regular hexagons.

Irregular tessellations are subdivisions of the plane into cells varying in shape and size. They are well suited for representing irregularly sampled fields. Nodes of their polygons can be located at the exact positions of samples, preserving the original values in the model. Due to the various sizes of cells they adapt to the complexity of the studied model, preserving all the details when necessary. This is different in grid models, which have fixed resolution and do not adapt to the density of samples, so the cells size is crucial in preserving details of the model. The attribute inside each cell can be a constant value or a function, usually linear. In GIS the two most popular examples of irregular tessellations are choropleth maps (Figure 2.3) and triangulations.

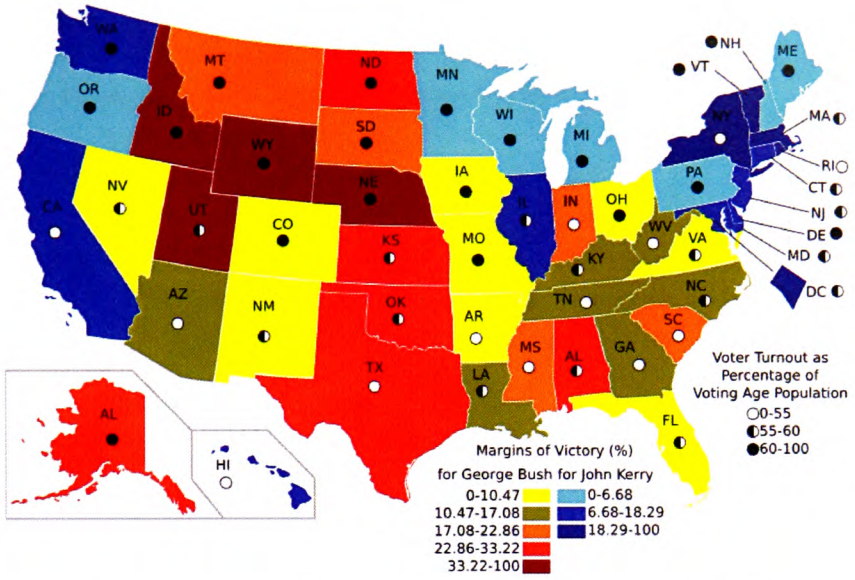


Figure 2.3: An example of a choropleth map. (from Wikimedia Commons)

Choropleth maps consist of irregular polygons with a constant attribute attached to each of them. They are mostly used to visualize statistical variables such as density or proportions. They are often drawn with a colour symbolizing the intensity of the variable and used to visualize how the measurement varies across the map. Their creation has been automated with current GIS systems. Practical examples of choropleth maps include maps of countries, population densities, per-capita income or presidential election results, as in Figure 2.3.

Triangulations subdivide the plane into adjoining triangles, which are the simplest polygons. Triangles are usually created by joining the locations of samples. The Delaunay triangulation, described in Section 3.3, is the most popular triangulation. Attributes inside each triangle are usually obtained by a linear interpolation on the plane passing through the three triangle vertices, although more complex functions can also be used.

2.4.2 Object Models

Unlike the field model where the attribute varies continuously over the domain, the object-based model of the world populates the empty space with entities (such as buildings,) that can be assigned to different classes (de Smith et al. (2008)). They are often man-made facilities (cars, buildings) or entities derived from field models by identifying regions or features (mountains, geological faults). They are spatially referenced and are described by their attributes. Attributes defining objects can be divided into three groups: referring to the location of the object (coordinates, elevation), describing non-spatial properties (name, ownership) or derived from the spatial properties themselves (area, length) (Burrough and McDonnell (1998)). They can also refer to time, stating for example when the object was created or modified (Worboys and Duckham (2004)). Objects can be represented as points, lines and polygons (also called "areas", consisting of connected lines) depending on their size in relation to the modelled space (Goodchild (2005)). Objects interact with each other - by overlapping, being adjacent or enclosing others and GIS systems must provide mechanisms to select, retrieve and analyse them. When operating on objects in a GIS they can be "counted, moved about, stacked, rotated, colored, labelled, cut, split, sliced, stuck together, viewed

from different angles, shaded, inflated, shrunk, stored, and retrieved, and in general, handled like a variety of everyday solid objects that bear no particular relationship to geography" (Couclelis (1992)). Figure 2.4 shows a set of objects representing buildings extracted from a complex city map.

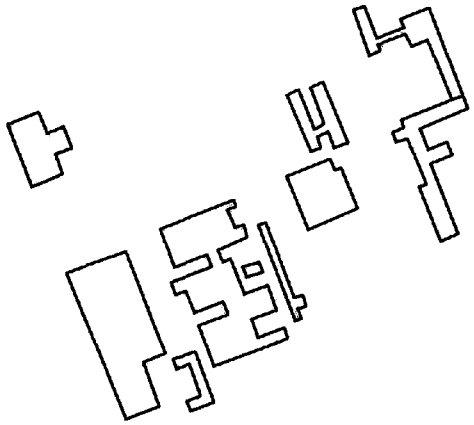


Figure 2.4: Buildings as discrete objects.

2.4.3 Integrating Fields and Objects

In recent years there have been many discussions about the object-based and field-based conceptualizations and models, and a significant amount of research conducted on their integration. In one of the earliest papers Couclelis (1992) argued that the field/object distinction was fundamental to humans understanding of the world, as we make use of both the object and field views. We perceive the world surrounding us as populated by discrete objects, while many properties of the environment are perceived as continuous fields.

Winter (1998); Winter and Frank (2000) presented a hybrid representation model for the raster and vector representation models using a strict mathematical formalism. It is based on a different interpretation of a raster in which each cell is closed by four edges and four nodes. Such a union of edges and nodes enclosing cells is called a "skeleton". They claimed that extending the raster with its skeleton assimilates the behavior of vector and raster representations. Using this model it is possible to construct an integrated GIS, in which one set of operations can be applied to data, independent of its format. In our model grids can be represented in various ways, including using Voronoi cells with vertices at the locations of the grid nodes (as described in Section 6.1.5) or a line segment VD with line segments along edges of grid cells. Such converted grid models can be analysed then in the same way as other Voronoi diagrams.

Cova and Goodchild (2002) linked field and object representations of geographical space using a typology of an object field. Locations in a field space are related to objects in an object-space, so space can be modelled as relationships between field and object perspectives. In their multi-representational perspective each location in a continuous field points to any number of discrete objects (one to zero, one to one and one to many relations). The object field extends the traditional field where every location determines a scalar, vector, or tensor to one where every location is mapped to a discrete, geo-referenced object, so for example every location on the topographic surface can point to the area visible from there. Object fields can be represented with any combination of existing data models for representing fields and spatial objects. In our common

data structure for fields and objects we assume a one to one mapping of objects to fields producing tessellations with a dual representation available.

Kjenstad (2006) using Unified Modelling Language (UML) proposed a common base-model for the object-based and field-based conceptual GIS models, called the PGOModel or "Parametrised Geographic Object Model". Fields and objects are special cases of the base model and can be mapped to each other after some adjustments. The concept of PGOModel is developed to bridge two models of GIS, just like the hybrid model of Winter (1998), but they are different and developed using different tools. The object fields model of Cova and Goodchild (2002) can be treated as a special case of the PGOModel.

Goodchild et al. (2007) demonstrated that both object and field conceptualizations can be derived from a common fundamental "geo-atom" structure. It is an association of the point location and a property and can be written as a tuple $<x,Z,z(x)>$, where x is a point in space–time, Z identifies the property, and $z(x)$ is the particular value of the property at that point. Both fields and objects aggregate the locations that are the first element of the geo-atom tuple. A geo-field is an aggregation of geo-atoms defining the same set of properties (the second element of the geo-atom tuple); while a geo-object is formed by aggregating geo-atoms according to rules defined on the third element of the tuple, the geo-atom's value. Conversion between them is essentially an interpolation problem. Additionally, they also introduced the concept of a geo-dipole as "a tuple connecting a property and value not to one location in space–time as in the case of the geo-atom but to two: $<x1,x2,Z,z(x1,x2)>$". "Geo-dipoles capture the properties of pairs of points, or properties that are associated with two points rather than one." "Geo-dipole provide a foundation for a set of concepts dealing with such properties as distance, direction, interaction, and flow". They describe the six different and most popular representations of fields (the same as in Goodchild (1993)): three of them are complete, and three others are not. However, they do not describe the proximal case, nor the case of non-point objects. By contrast, we use the same proximal model based on the VD for all point, line and compound cases.

Liu et al. (2008) believed that the field view is more fundamental than the object view, since the field view represents a stage before the identification of discrete features and object models have a higher level of abstraction and it is reasonable to combine objects and fields at the lower abstraction level, thus managing objects using fields. They proposed a concept of the General Field (G-Field) by treating a set of objects as an object field and unifying it with traditional field models. They showed that fields and object-fields can be seen as specialized G-Fields. Discrete objects are considered a special case of an object field in which each point maps to the identity or attributes of any object in which it lies, or else to zero value. They also showed that G-Fields can be implemented in GIS using one of the three known methods: sampling and interpolation-based field, tessellation-based field, and real-time generated field; and that geographic analysis can be perceived as a sequence of operations on G-Fields.

2.5 Raster and Vector Formats

Fields and objects are ways in which geographical phenomena can be thought of and are not designed to be based on particular computer representations. Fields conceptually contain an infinite amount of information, as the attribute value can be extracted at any location. Objects also need an infinite numbers of points to precisely define their boundaries. There are two common ways of representing fields and objects in computers - in vector or raster formats (Longley et al.

(2001)). There are many differences in these two representations, but the primary theoretical difference is that "the raster structure stores information on the interior of areal features, and implies boundaries, whereas, the vector structure stores information about boundaries, and implies interiors" (Berry (1993)), so for example a lake can be represented by a set of cells represented its interior or by connected lines defining its boundary. Objects are usually represented in vector formats, while vectors and rasters are equally common for fields.

Phenomena in vector format are depicted using point, line and polygon features referenced by Cartesian coordinates (Burrough and McDonnell (1998)). Each feature can be organized into a data structure and stored as a record or an object, containing information about the location, attributes or topological relations between features.

Points are the primary components of vector models. The main information stored in a point record is the information about its location in the coordinate system. Besides the georeference, additionally there is often extra data stored, like an identifier or attributes. Points are used to represent locations of wells, elevation spots or shops, etc.

Lines are used to represent one-dimensional objects, such as roads, streams and geologic faults. In the simplest form they consist of references to two points. More complex representations can have attributes and store pointers to other lines. Line objects linked by special kinds of points, called nodes, form "strings" or "arcs", used to model networks and polygons and to approximate curvilinear features. Nodes store information about how the lines are connected and can be used to model traffic flow between them (Burrough and McDonnell (1998)).

Polygons are used to represent geographical phenomena having defined boundaries. They can be stored in various ways. In the simplest form a polygon is represented as a sequence of lines defining its boundary (spaghetti model). Such a representation is easy to create and store, but common boundaries of two polygons are stored twice (data redundancy), they might slightly overlap and lack of connectivity information means that traversing or adjacency operations require additional calculations. In more a complex representation a polygon can be represented by a sequence of lines with indications what polygons are located on the left and right side (de By (2001)), so each boundary segment is stored once (no overlaps between adjacent polygons) and the information about neighbouring polygons (topology) is available. When representing fields non-overlapping polygons cover the whole space (planar enforcement), while in object models there can be empty space between polygons. Triangles are the simplest polygons and they are the components of TIN models often used to represent terrain, while complex irregular polygons are used to build choropleth maps.

The raster divides the plane into an array of equal sized cells, usually squares (Longley et al. (2001)). Each cell references a square parcel of the Earth's surface. The resolution of the raster is a relation between the size of the parcel and the size of the grid cell. Grid models can be nested to provide more detail using quad-tree structures, as described later in this section.

The variation of the phenomenon is stored in grid cells as their attributes, e.g. integer numbers in Figure 2.5. In the simplest form the attribute is a binary value, for example when modelling the presence or absence of a certain feature. In more advanced applications it can be a floating point number, for example in elevation models. Also multiple attributes can be used and accessed, by means of pointers in each cell referencing the information stored in tables or databases (Burrough and McDonnell (1998)).

7	7	7	6	6	7	7	8
8	7	7	7	7	7	8	8
8	8	7	7	7	8	8	8
8	8	8	8	8	8	8	8
9	8	8	8	8	8	8	8
9	9	8	8	8	8	8	9
10	9	9	9	9	9	9	9
10	10	9	9	9	9	10	10

Figure 2.5: A grid with integer attributes.

Rasters are stored in various file formats, from common image TIFF or JPG files (different compression) to specific grid data formats. The raster files often start with header data at the beginning, containing the geographic coordinates of its corners, the size of the cell and the number of rows and columns. The actual data after the header can be stored in various ways, usually as a string of values, often using compression techniques.

To overcome problems with handling large grid files, a region quad-tree structure, which is a hierarchical tessellation, is often used. The quad-tree subdivides recursively a non-homogeneous array of squared cells into four cells of equal size, until there is homogeneity in each cell (Worboys and Duckham (2004)). The structure is implemented in computers as a tree with each node having four (if this cell is divided) or zero (final, not divided cell) children. There are different types of quad-trees and the most relevant for modelling fields is the PR quad-tree (Point-Region), which for a given set of points divides recursively the plane into four cells of equal size, until each cell contains only one point (van Oosterom (1999)). The biggest advantage of a quad-tree structure is its variable resolution. However, it is not very efficient in representing objects composed of points and lines and highly irregular features, as they require a large number of local cell subdivisions producing deep and unbalanced trees. Also, even a small translation or rotation of the objects can produce a considerably different quad-tree structure (Worboys and Duckham (2004)).

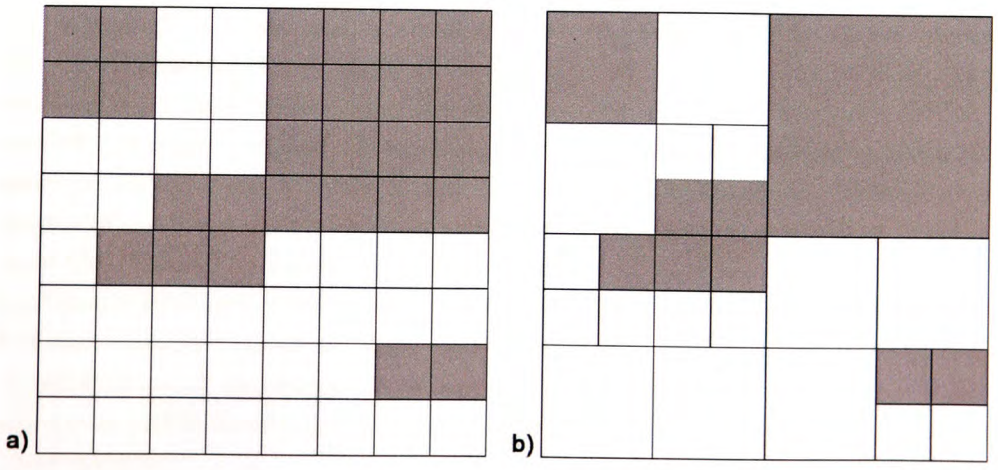


Figure 2.6: A region quad-tree subdivision for a grid. a) The fixed resolution grid. b) The resulting quad-tree. (after Ledoux (2006))

Vector and raster formats have both many well known advantages and disadvantages (see

Burrough and McDonnell (1998) p.70 for a discussion). The popularity of rasters is mostly due to the simplicity of their construction and analysis. Their structure fits the traditional way of storing data in arrays. Operations like overlay are trivial using grid data if input grids are aligned. However, grids are not space efficient and often large grids are needed to represent information (although they can be compressed or quad-trees can be used). Since the resolution of grids is fixed they do not adapt to the data distribution. Using larger cells decreases the storage cost, but also decreases the resolution of the model and leads to loss of information. Grids do not represent boundaries well, because squared cells produce "blocky" approximations of linear features only. Also rotation and scaling operations are complex - rows and columns of pixels provide limited enlargement possibilities and make rotations problematic. As a result their popularity in some areas, like terrain modelling, has been decreasing steadily in favour of triangulations.

The structure of vector models is more complex than rasters, but usually they use less space than grids for storing the same data. Vectors adapt to the data distribution and each input data can be exactly presented as a node in the model. The elements in vector models can be of any shape which makes them better suited for representing linear features than grids (with their blocky "lines"). Vector models are easy to maintain and modify. Scaling and rotating vectors is also much easier as these are just vector operations. Also in vector models the topology (relationships between objects) can be defined explicitly by pointer links between objects. In grids the only topology available is the cell adjacency and it is defined implicitly by the grid address, as cells are addressed by the row and column number. These arguments prove that a vector representation of the data is better choice for GIS.

2.6 Spatial Analysis

The aim of GIS is not only to store geographical phenomena in a digital form, but also to provide opportunities to operate on them. Spatial analysis, being a collaboration between the human and the computer, has been defined by Longley et al. (2001) as "the crux of GIS, the means of adding value to geographic data, and of turning data into useful information" and it "includes all of the transformations, manipulations, and methods that can be applied to geographic data to add value to them, to support decisions, and to reveal patterns and anomalies that are not immediately obvious". Spatial analysis results depend on the information about the location - its results change when the location of the analysed object changes.

Most GIS operations or classifications (taxonomies) depend on the underlying vector or raster data structure, so not every operation is possible for each of the structures. There are very few existing general classifications and two most known are the map algebra proposed by Tomlin (1983) and the 20 GIS universal operators by Albrecht (1996).

Burrough and McDonnell (1998) classified operations separately for objects and fields and listed groups of most common operations. Raster models were used to classify operations on fields and it was noted that not all operations can be used in vector models.

Longley et al. (2001) divided spatial analysis methods into six groups, ranging from the simplest to the more complex:

- Queries and reasoning - The most basic operations, where the GIS simply answers queries from users. The user interacts with the system by pointing at the map or selecting operations from the menu, which sends a request to the database. No data is modified or added.

Operations include object selection, attribute retrieval and proximity queries. The term of "reasoning" is interpreted as a collection of methods to respond to more complex queries allowing vague terms, including navigation.

- Measurements - Measurement of simple properties of objects, including length, size, shape, area, slope and aspect, and relationships between them, like distance or direction.
- Transformations - Creation of new objects by transforming database information or objects. Examples include buffering, spatial interpolation, point in polygon and polygon overlay operations.
- Summary statistics - Summarizing various properties of GIS data, like centres, dispersion, histograms and pie charts, scatter-plots.
- Optimization - Optimizing specified objectives, applications include optimal point location, routing on a network, optimum paths on a surface.
- Hypothesis testing - Process of making generalization about the entire population using a limited set of samples.

Albrecht (1996)presented a set of 20 universal analytical GIS operations separated into six categories. They are independent of data structures and "allow to build all but the most exotic GIS applications". He represented a task-oriented user's point of view, instead of a more common developer's view and his operations do not require a special knowledge about spatial concepts.

- Search - interpolation, thematic search, spatial search, (re)classification.
- Location analysis - buffer, corridor, overlay, Thiessen/Voronoi.
- Terrain analysis - slope/aspect, catchment/basins, drainage/network, viewshed analysis.
- Distribution/Neighbourhood - cost/diffusion/spread, proximity, nearest neighbour.
- Spatial analysis - multivariate analysis, pattern/dispersion, centrality/connectedness, shape.
- Measurements - various measurements, such as distance and area.

Dana Tomlin's Map Algebra (Tomlin (1983), Goodchild (2002), Camara et al. (2005)), also known as cartographic modelling language, is based on the raster data structure and uses math-like expressions with operators and functions. Each field of the raster is treated as a variable and operations on fields form sequences similar to equations. It is "the first significant attempt at achieving (...) separation of operations from consideration of the form of the spatial discretisation." (Kemp (1993))

There are three groups of operations, depicted in Figure 2.7:

- Local operations – the value of a cell at location x, y in one layer is based on values of cells at the same location x, y in other layers, can be used to compute the minimum or the average value of pixels in all layers at x, y.
- Focal operations – the value of a cell at location x, y is computed using specified neighbouring cells, can be used for smoothing data or calculating slope in terrains.

- Zonal operations – the value of a cell at location x, y is computed using values of cells from another map, can be used to for operations such as "given a map of cities and a digital terrain model, calculate the mean altitude for each city" (Camara et al. (2005)).

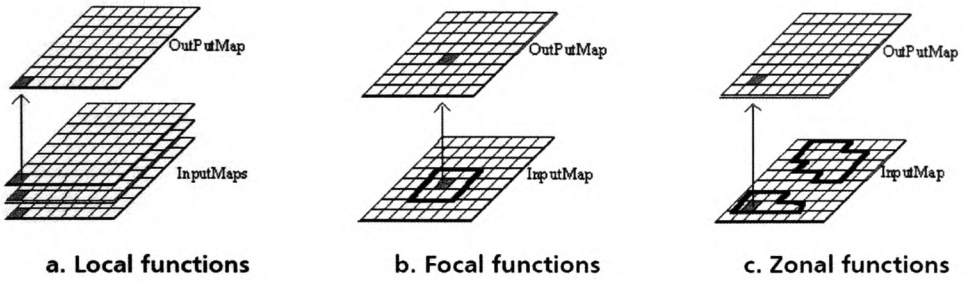


Figure 2.7: Tomlin's map algebra. (from Camara et al. (2005))

There are some well known drawbacks of Tomlin's Map Algebra. It is restricted to fields stored in raster format (which has many limitations). The scale and orientation of the rasters must be the same, so one cell in one raster corresponds to one cell in another raster. Recently Ledoux and Gold (2006) proposed a Voronoi-based Map Algebra, where they perform all operations on the Voronoi diagram instead of regular tessellations.

2.7 Problems with Traditional GIS Structures

GIS is concerned with the manipulation and analysis of spatial data. Apart from issues of storage, database query and visualization, they must deal with several significantly different types of spatial information, such as networks, polygons or terrains. Each of these has a specific set of assumptions associated with it, a specific data structure, and a specific set of algorithms. Details of data structures and algorithms used in commercial GISs are not usually available and based on the recent textbooks it appears that different structures and methods are used for different types of data. Spaghetti, DCEL or edge based structures are used to model objects and polygonal maps, various edge based structures to model networks and grids, TINs or Voronoi diagrams to model terrains (Worboys and Duckham (2004)). This produces a high level of complexity in the construction, manipulation, analysis and comparison of these datasets. It is possible to compare and perform operations on different types of layers, but there is no consistent method applicable to all data types. Often the same operation is performed differently depending on the structure of the layer. The simplest example is changing the location of an object. When performed in a discrete object layer, this means changing the object location attributes in the associated table in the database. The same operation performed on a polygon in a field model requires changing coordinates of the polygon and updating the topological information for the polygons adjacent to the old and new location of the object. Additionally, the lack of topology in discrete objects creates difficulty in some applications. Relocating a node of a network consisting of discrete line objects (described only by their coordinates) may lead to overlaps of lines and invalid graphs. This can be avoided using topological networks, where the connectivity is preserved when changing the positions of nodes (although without some collision detection mechanisms it may lead to unwanted intersections of edges). The lack of topological information in object based models also creates problems with queries. They can only report presence or absence of objects, and have no access

to adjacency information, so interpolation operations are difficult. Also, an object movement simulation needs the ability to detect collisions. This requires maintaining proximity information between objects, which is not present for discrete unconnected objects, but is readily available in topological field models (Gold (2006)).

This thesis reports the results of an attempt to integrate objects and fields based on a slight modification of the fundamental spatial query: "What is here?" where "here" is usually an (x, y) location. When "here" is replaced by "closest to here" the resulting proximal query (a Voronoi diagram) may be used to manipulate the four categories described above, with a resulting simplification of the system. All discrete objects become "fields", with a value at any location. The catch is that in order to represent non-point objects a line-segment Voronoi diagram is required, and this has been shown to be extremely complex to construct on digital computers. Nevertheless, with the availability of several potential solutions, including our own, we can show the implementation of a basic GIS with a common data structure and set of operations. Spatial analysis, which in GIS frequently consists of overlaying data sets to identify potential conflicts, may then be performed in a consistent and straightforward fashion.

Chapter 3

Computational Geometry

Computational geometry (CG) is a branch of computer science dealing with the design and analysis of algorithms for geometric problems and objects. It emerged in the late 1970s and the publication of Shamos' PhD thesis entitled "Computational Geometry" in 1978 is often cited as the founding event. Important applications of computational geometry include the fields of computer graphics, robotics, geographic information systems and CAD/CAM. Typical problems are object intersections, visibility, motion planning and object modelling. The rapid development of computer technologies and software in the last thirty years gave the research in CG a boost and it grew to an important field of computer science.

The Voronoi diagram (VD) and Delaunay triangulation (DT) are two of the most important tools in computational geometry and are used to solve many issues, such as proximity calculations or some graph operations. The VD has been known and used for more than 100 years. In 1850 a German mathematician Lejeune Dirichlet utilized the structure in his study of quadratic forms (Dirichlet (1850)) and now often the diagram is called the Dirichlet tessellation. Voronoi diagrams are named after Georgy Voronoi - a Russian mathematician who defined the n-dimensional general case of the diagram in 1908 (Voronoi (1908)). The diagram was first used in geography in 1911 by an American meteorologist Alfred H. Thiessen in his work on climatology (Thiessen (1911)) and is often called Thiessen Polygons by geographers. The Delaunay triangulation, the dual graph of the VD, was formalized in n-dimensions by Boris Delaunay in 1934 (Delaunay (1934)).

In practical applications it is often required that a pre-specified set of segments is included in the triangulation. There are two versions of Delaunay triangulations allowing that - the Constrained Delaunay Triangulation (CDT) where no additional nodes are added (developed independently by Lee and Lin (1986)and Chew (1987)) and the Conforming Delaunay Triangulation requiring additional nodes for preserving segments (Edelsbrunner and Tan (1992)).

The Voronoi diagram and Delaunay triangulation have applications in geography, physics, robotics, astronomy and many other disciplines. They have traditionally been used for solving distance problems (Aurenhammer (1991)). The classical example is the nearest neighbour problem, where for a set of n given sites the nearest site to a specified location x has to be determined, which means finding which Voronoi cell contains x. Another classical issue is the all nearest neighbours problem, where the nearest neighbour is reported for each of the sites from a set V, by testing the neighbours of each node and reporting the nearest ones. The result is a tree-like oriented graph being a subset of the DT of V, with an edge pointing from each site towards its nearest neighbour. A related problem of the closest pair of sites can be solved finding the shortest edge in the DT of

V. Also the largest empty circle and smallest enclosing circle of the sites can be found using the VD.

This chapter provides definitions of the VD and DT and other diagrams and concepts associated with them, that are used in next chapters.

3.1 Duality

Two graphs are dual if there is a one-to-one relation between their nodes, edges and cells (Boots (1999)). They represent the same thing, but from a different perspective. A dual graph can be created in $\mathbb{R}^2$,by selecting one point in each cell and connecting it with selected points in adjacent cells. As we can see in Figure 3.1 each dotted edge corresponds to a solid edge, each vertex of the dotted graph corresponds to a polygon of a solid graph and each polygon of the dotted graph corresponds to a vertex of the solid graph. A different method of building a dual graph is to make a dual edge for each edge e of the graph, so endpoints of the dual edge do not need to be inside cells adjacent to the edge e.

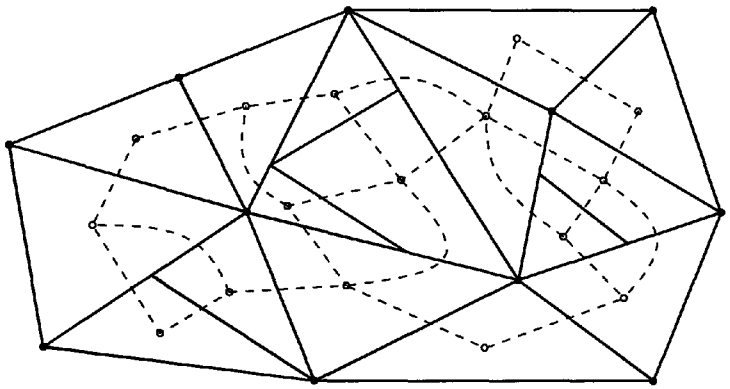


Figure 3.1: Dual graphs.

3.2 Voronoi Diagram

Let us start with a simple example. We have to assign children to schools. Usually the simplest idea coming to our mind is to send a child to the nearest school. Such a model, where every point is assigned to the nearest site is called the Voronoi assignment model (de Berg et al. (1997)). The resulting partition is called the Voronoi diagram.

More formally, the Voronoi Diagram of a set V of n vertices in a 2D plane $\mathbb{R}^2$, defined *VD(V)*, is a partition of R into n cells (Voronoi cells), one for each vertex of V, with the property that a Voronoi cell of a vertex $p \in V$ contains all points of the plane that are closer to p than to other vertices of V. The definition of the VD can be easily expanded to more dimensions than 2D.

A continuous set of points shared by two adjacent cells is called a Voronoi edge (so each Voronoi edge is equidistant to exactly two vertices) and a point shared by at least three Voronoi cells is called a Voronoi vertex. A Voronoi vertex is a circumcentre of the circle passing through three or more (when more than three points are cocircular) adjacent points. A Voronoi cell is defined by its Voronoi edges and Voronoi vertices. Figure 3.2 shows the Voronoi diagram of a set of points with two Voronoi vertices and circles defining them marked.

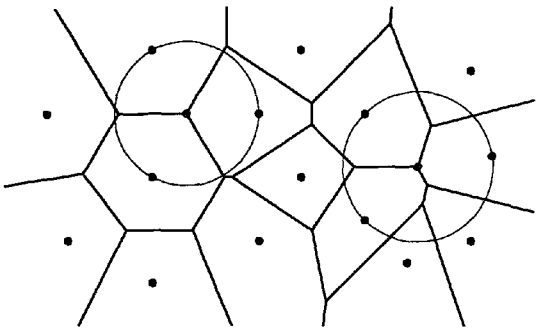


Figure 3.2: The Voronoi Diagram.

A Voronoi cell is bound by *n-1* edges at most. If there are two vertices only, then the VD consists of a line formed by points of the space that are equidistant from those two vertices. Such a line, called a bisector, separates the plane into two open half-planes. If there are more than two vertices then the Voronoi cell of p is an intersection of all half-planes containing p.

The Voronoi diagram can be constructed directly using incremental, sweep or divide and conquer algorithms (Aurenhammer (1991)). However, it is usually extracted from its dual graph – the Delaunay triangulation, by calculating the circumcentres of triangles (Voronoi vertices) and joining neighbouring centres with straight lines (Voronoi edges).

3.3 Delaunay Triangulation

A triangulation of a set of vertices connects them into non-intersecting triangles. There are many ways of triangulating a set of points and the Delaunay triangulation is the most widely used. Another example is the greedy triangulation (Dickerson et al. (1997)), minimizing the total edge length, by selecting the shorter diagonal at each step of the construction.

The Delaunay triangulation of a set V of n vertices, *DT(V)*, is a dual graph of the Voronoi diagram of V. It consists of triangles (Delaunay triangles) formed by edges (Delaunay edges) connecting centres (Delaunay nodes) of adjacent Voronoi cells. Figure 3.3a shows the DT (solid lines) and its dual VD (dashed lines). The DT and VD are dual graphs, thus Delaunay nodes correspond to Voronoi cells, Delaunay edges correspond to Voronoi edges and Delaunay triangles correspond to Voronoi vertices (Aurenhammer (1991)).

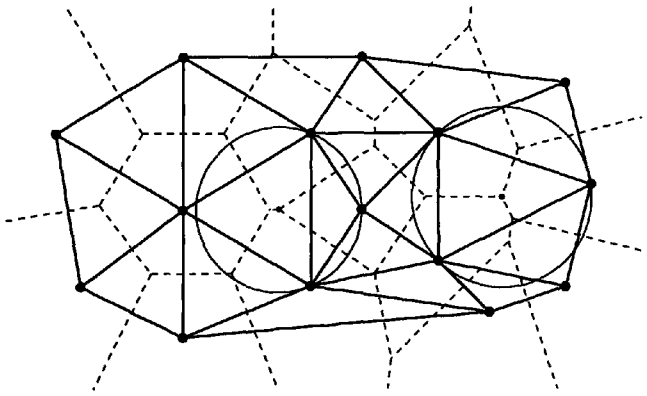


Figure 3.3: The Delaunay triangulation (solid lines) and Voronoi diagram (dashed lines) with two of the empty circumcircles marked.

Another practical and widely used DT definition is the one using the empty circumcircle property: a triangulation is Delaunay if the circumcircle of each triangle is empty. A triangle is Delaunay if its circumcircle is empty. There are two circumcircles marked in Figure 3.3.

The DT has many properties. Each of the edges has a point-free circle passing through its endpoints and each triangle's circumcircle is point-free. The DT maximizes the minimal angle of each triangle (the Min-Max angle property). Delaunay edges are perpendicular to corresponding Voronoi edges, but do not necessarily intersect them. Numerous existing techniques used to construct Delaunay triangulations of points, including batch methods and incremental algorithms, are presented later in Sections 4.2 and 4.3 of this thesis.

The DT of a set V of vertices $DT(V)$ contains various graphs as subgraphs. One of the most important is a Minimum Spanning Tree $MST(V)$ of V, which is a connected graph that has a minimum total edge length (Aurenhammer (1991)). The MST has many applications for solving transportation problems, clustering and pattern recognition. Also a travelling salesman tour, $TST(V)$, which is a minimal length cycle passing through all nodes of V, is a subset of $DT(V)$. The DT can also be used to extract α-shapes and β-skeletons (Amenta et al. (1997)). For a specified value of α, the α-shape $\alpha(V)$ of V contains an edge connecting two nodes u and v of S if there is a circle of radius α passing through u and v and is empty of other nodes of V. For sufficiently large α the convex hull of S is obtained. A decreasing α increases the resolution of the shape and when α is smaller than the shortest edge of $DT(V)$ then the α-shape contains no edges.

3.4 The Voronoi Diagram of Moving Points

The Voronoi Diagram of Moving Points, also called the Kinetic Voronoi Diagram, is one of the modifications of the original VD, allowing changes in the positions of nodes over time. The simplest way of changing the position of a vertex is to remove it from the diagram and reinsert it at the desired location. Using such an approach to simulate movement of a node would involve consecutive insertions and deletions of the node at intermediate locations along the specified trajectory. However this method is not efficient and very expensive computationally. The alternative approach is to "drag" the vertex in the diagram while maintaining topological relationships with its neighbourhood. Roos (1993) noticed that in opposition to a Voronoi diagram which changes constantly as its nodes moves, the dual Delaunay triangulation is locally stable for sufficient small continuous node repositioning. So the idea is to use the DT for movement simulation and update the VD when necessary (Gold (1990)).

Assume we are given a set of nodes continuously moving along specified trajectories, which are straight lines connecting start and end points of the movement. At any instant in time the Voronoi Diagram expressing the relationships between them can be defined (Albers et al. (1998)). As the points move the diagram changes continuously and at certain moments "topological events" (cocircularity events) occur changing the neighbourhood relations of nodes. Those events can be observed in the dual Delaunay Triangulation graph as flips of edges (which means switching of the diagonal edge of a quadrilateral, as in Figure 3.4). Topological events occur when four points (defining two adjacent triangles) become cocircular (Gavrilova and Rokne (1999)), so circumcentres of their triangles are in the same location (the length of the dual Voronoi edge of the quadrilateral's diagonal is zero) and the VD is the same for both configurations of the quadrilateral, as shown in Figure3.4. The empty circumcircle test run for any of those quadrilateral configurations reports that the circle is not empty, as there is a point located on that circle. This forces edge swapping

and is a key component of the moving process.

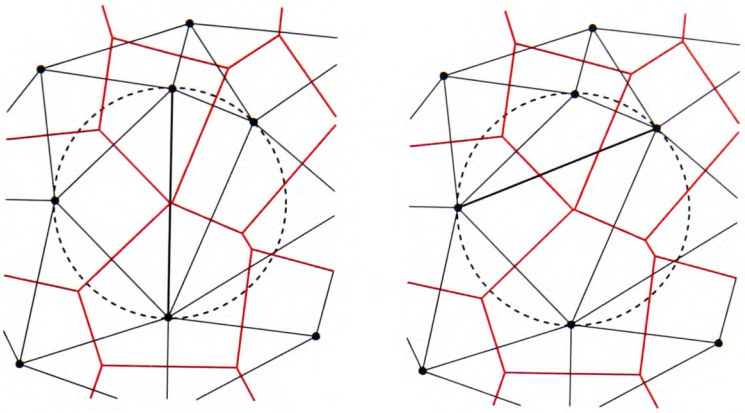


Figure 3.4: Two configurations of the quadrilateral with four Voronoi cells touching at the same point.

Movement of a point does not require changes in the topology of the VD as long as the point does not become cocircular with its neighbouring vertices. Figure 3.5 shows a moving point *MP* and circles affected by movement are marked in gray or dashed lines. Gray circles are circles that *MP* can enter (circumcircles of neighbouring triangles), and dashed ones (circumcircles of "imaginary triangles" formed by consecutive triples of neighbours of *MP*) are those that *MP* can leave. These will be described later in detail in Section 4.5. The white area is an area where *MP* can be moved without triggering any topological events, so it stays connected to the same set of neighbours and only the shape of its Voronoi region changes, while entering the gray area requires some changes in the topology of the mesh (flipping edges).

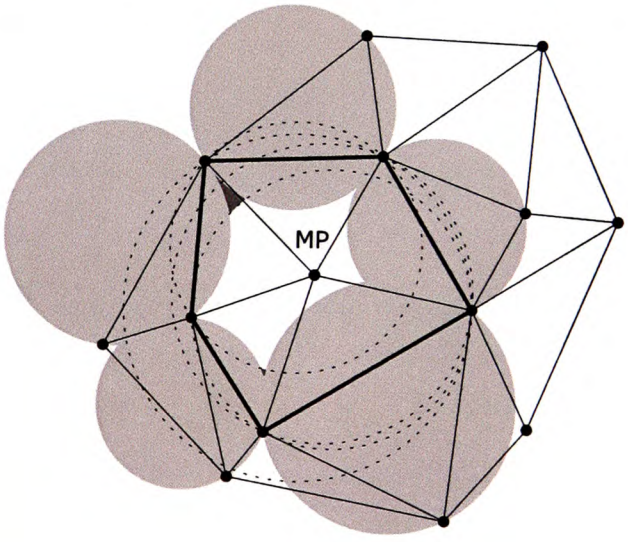


Figure 3.5: The moving point P and an area in which it can be moved without changes in the topology of the VD. (after Ledoux (2006))

Figure 3.6 shows three consecutive steps of moving a point *MP* vertically upwards. In Figure 3.6a *MP* is in the initial position, outside the circle. In the next step *MP* is moved onto the circle *abc* and *MP* becomes cocircular with points *a*, *b* and *c*. The shapes of Voronoi cells adjacent to

MP are recomputed and the length of the dual Voronoi edge of *ac* is reduced to zero. Then the diagonal edge *ac* of the quadrilateral is swapped, joining *MP* with *b*. In the next step *MP* is moved further towards *b*, entering interior of the circle. The distance between *MP* and *b* is reduced, the length of the dual *MPb* edge increases and the sites *a* and *b* are clearly separated. The same set of figures in the opposite order illustrates a movement of *MP* downwards, as it leaves circle *abc*, losing *b* as its neighbour.

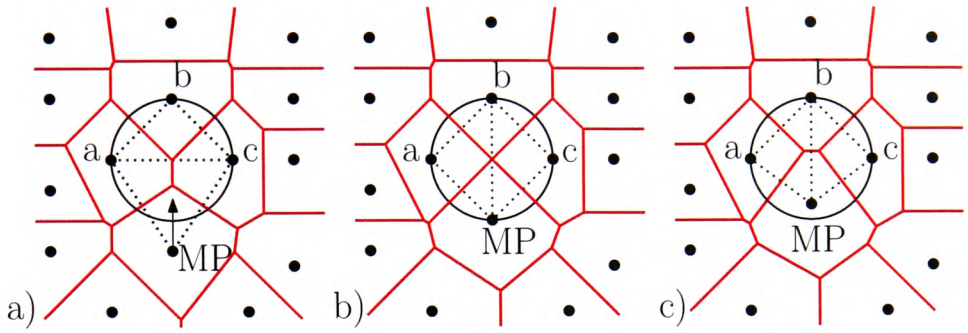


Figure 3.6: A moving point MP sequence. a) MP before entering the circle. b) MP on the circle. c) MP after entering the circle.

3.5 The Constrained Delaunay Triangulation

Assume we are given a set of nodes V and segments S with endpoints among V. A constrained triangulation is a triangulation of V containing all segments of S among its edges. It can be created by connecting nodes of V and endpoints of segments S with as many straight edges as possible without crossing any of the edges or segments.

The constrained Delaunay triangulation (CDT) (also known as a generalized Delaunay triangulation (Lee and Lin (1986))) possesses one more property – it is as close as possible to the Delaunay triangulation (Chew (1987)). Figure 3.7a shows an arbitrary constrained triangulation of a set of segments and nodes created by connecting them without any criteria except that edges cannot intersect. Figure 3.7b shows the same set of points and constraints triangulated using Delaunay properties.

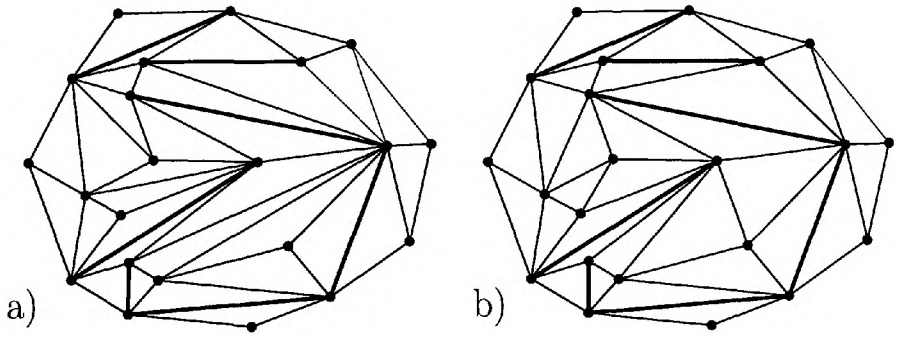


Figure 3.7: Constrained triangulations. a) An arbitrary constrained triangulation. b) A constrained Delaunay triangulation.

The constrained Delaunay triangulation is often defined using the visibility property. Two points are visible from each other in a set of segments S if a line connecting them does not cross

or touch any of the segments of S. Using this notion we can define a triangle as a constrained Delaunay triangle if its interior does not intersect any of the input segments and its circumcircle does not contain any vertices that are visible from that triangle. If there is a vertex inside the circumcircle it must be occluded by another constrained edge so it cannot be connected with any of the vertices of the triangle without intersecting other constrained edges (the vertex and the triangle lie on opposite side of that constrained edge). Figure 3.8 shows a constrained DT with three constraints. The marked circumcircle contains some points inside, but they are not visible from the triangle defining that circle so the triangulation is constrained Delaunay (because true for other triangles).

The empty circumcircle definitions of the CDT and DT are almost the same, with one modification that in the CDT portions of circumcircles crossing segments of S are ignored (Chew (1987)).

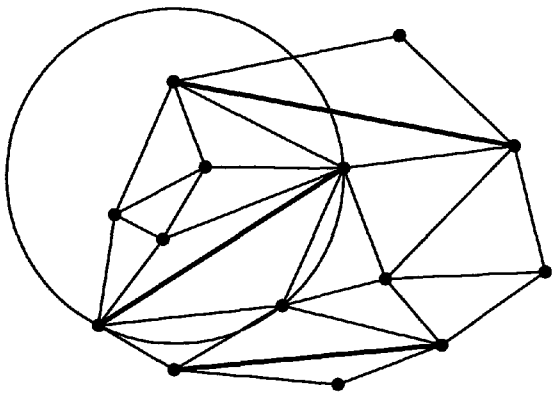


Figure 3.8: The visibility property in CDT.

The Constrained Delaunay triangulation in 2D can be used to extract a Voronoi diagram graph as well. A Voronoi diagram for a set of segments S can be defined using a visibility criterion (Chin and Wang (1999)). It is a subdivision of the plane into cells associated with endpoints V of segments S, such that a point p belongs to the cell of vertex v if v is visible from p. Such a diagram is not a dual graph of the CDT as some of the dual edges might not be present. We can notice that in Figure 3.9a where a dual edge of the constrained edge is not present in the diagram, nor the two Voronoi vertices of both triangles adjacent to that edge. The proper dual graph of the CDT is called the extended Voronoi diagram (also known as constrained or bounded) and was introduced by Seidel (1988). It exists on a surface more complex than the two-dimensional plane (Edelsbrunner (2000)) with half-planes attached to both sides of each of the cut open constrained edges, extending Voronoi edges beyond both sides of the constrained edges.

If we extract the VD from the CDT using the methods used for Delaunay triangulation (calculating and assigning Voronoi vertices for every Delaunay edge) we often end up with overlapping Voronoi cells, for example for triangles adjacent to long constrained edges. This is presented in Figure 3.9b, where circumcentres of triangles adjacent to the long constrained edge are on the opposite sides of that edge and Voronoi cells overlap. For simplicity of their calculations such diagrams are used in most of this thesis.

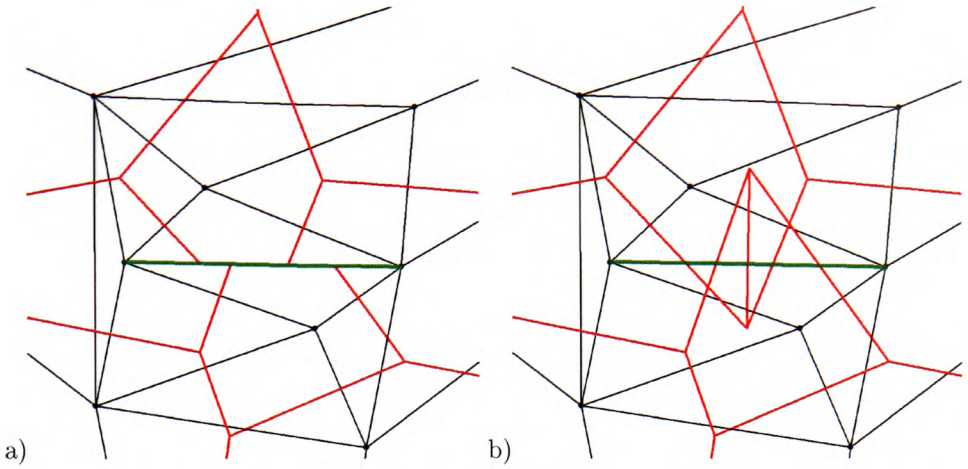


Figure 3.9: The Voronoi diagram of a CDT a) calculated applying the visibility criterion; b) calculated using the ordinary DT method (the thick edge is constrained).

3.6 The Conforming Delaunay Triangulation

The conforming Delaunay triangulation is another way of triangulating segments. Basically it is a constrained Delaunay triangulation having all Delaunay edges. All input segments are present and each of them is represented by a linear sequence of Delaunay edges. A Delaunay triangulation conforms to a set of segments S if every segment of S is present in the triangulation as a single edge or a set of edges (Edelsbrunner and Tan (1992)), all satisfying Delaunay criteria.

Any constrained Delaunay triangulation can be converted into a conforming Delaunay triangulation by insertion of additional vertices (Steiner points) on constrained edges until they are cut into Delaunay edges (Boissonnat et al. (1988)). Edelsbrunner and Tan (1992) shown that for any planar straight line graph with n nodes and m segments a number of m^2n nodes is sufficient to build a conforming DT of that graph. The minimum number of Steiner points is hard to predict and often the configuration of input segments (a short distance between them or sharp angles at junctions) forces a large number of them to preserve the Delaunay criterion. Bern and Eppstein (1995) discuss various techniques of Steiner point selection and the conforming DT construction.

The Voronoi diagram of a conforming Delaunay triangulation, unlike for the CDT, is the ordinary Voronoi diagram.

Figure 3.10 shows different triangulations of the same set of constraints (together with corner points), being line segments with endpoints centered in the same point (Figure 3.10a). Figure 3.10b shows the Delaunay triangulation of the endpoints and none of the constraints is present. In Figure 3.10c the constraints are incorporated into the triangulation using the conforming DT approach with some additional Steiner points added along the input segments. Figure 3.10d shows the CDT of the input constraints. All the triangulations are very different and depending on the application any of them can be used.

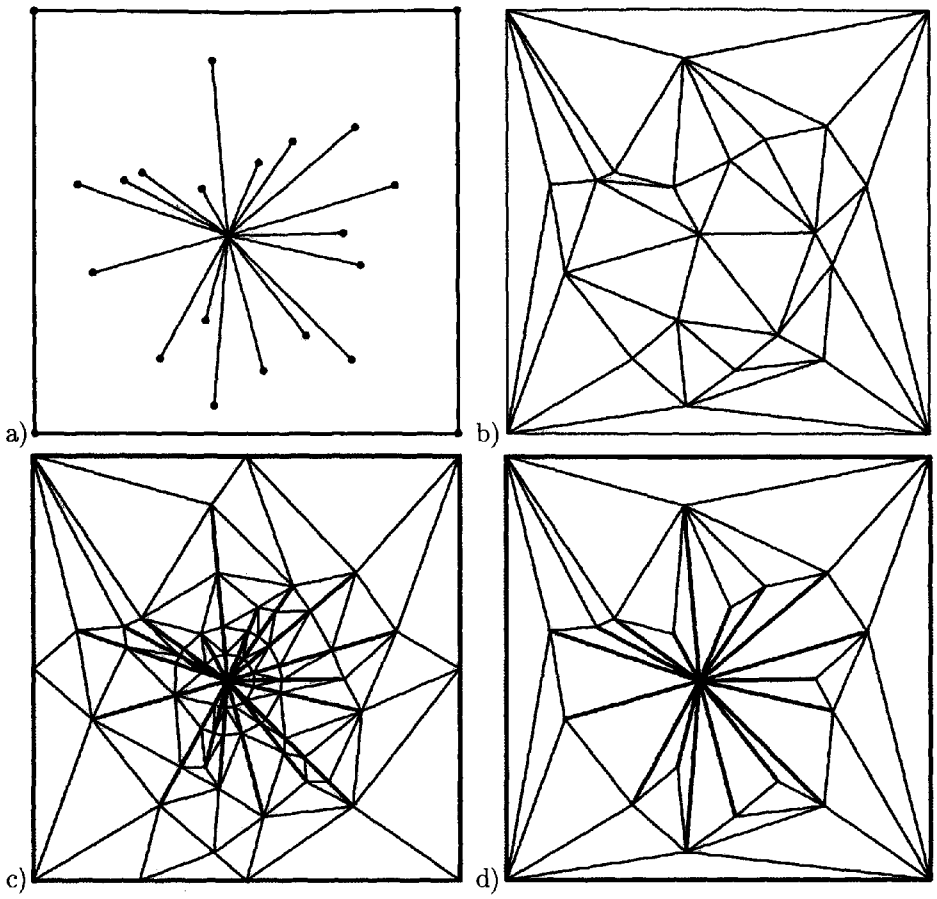


Figure 3.10: Conforming DT vs constrained DT. a) A set of constraints. b) The DT of the endpoints of constraints. c) A conforming DT (one of the solutions). d) The constrained DT. (from Shewchuk (1998))

The advantage of the constrained DT over the conforming DT is that it does not need any additional points to represent the constraints so the storage cost is usually lower. However, distinguishing between ordinary and constrained edges requires an additional mark in the data structure, or other processing. The biggest advantage of the conforming DT is that it has all Delaunay properties (although the CDT retains some useful DT properties as well). Also the shape of the triangles is usually more regular than in the CDT where thin triangles adjacent to long constrained edges often exist.

3.7 The Line Segment Voronoi Diagram

The main problem for using the Constrained DT in GIS is confusion of entities. Geometrically and topologically there is no difference between constrained and ordinary triangulation edges, and the only thing that makes them different is the value of the "constrained" attribute. The attributes and topological information are all "mixed" in the structure of the edge, which is not an elegant solution. Additionally, in the CDT the Voronoi regions are not associated with constrained edges but with nodes only and some of the Voronoi regions are not complete (due to the visibility property). So when constrained edges represent polygons, the Voronoi regions of their endpoints they often do not form proximal regions of these polygons. There exists another way of incorporating segments

into the triangulation, by treating them as objects, in the same way as vertices, so objects are separated from topology and additionally proximal regions are available for them.

The Voronoi diagram for line segments is a generalization of the ordinary Voronoi diagram, in which points are extended to straight segments. It can be defined in the same way as the VD for points, adding segments to the input data set. Given a set of generators $G=(V,S)$ where V are vertices and S are straight line segments, the Voronoi Diagram of G partitions the plane into cells with the property that a Voronoi cell of a generator $g \in G$ contains all points of the plane that are closer to g than to other generators of G (Gold (1990); Gold et al. (1995)), see Figure 3.11a. For simplicity the LSVD abbreviation will be used for Voronoi Diagram for points and line segments.

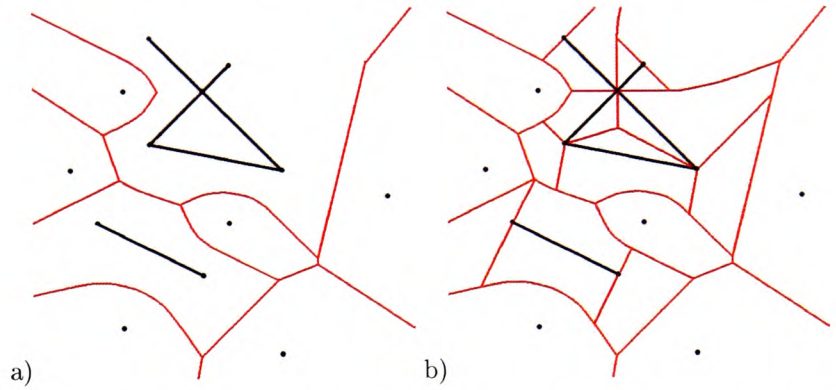


Figure 3.11: Voronoi diagram for points and segments generated for a) not decomposed segments; b) decomposed segments.

A common technique is to represent line segments as decomposed segments. Each segment is built not by one object, but by three - an open segment and two vertices - its endpoints (Imai (1996)). Each of the three elements has its own Voronoi cell. Figure 3.11b shows a resulting Voronoi diagram for points and decomposed segments.

There are four kinds of bisectors in the LSVD of decomposed generators (Okabe et al. (1992), Figure 3.12). They can be straight or parabolically curved edges, depending on the configuration of neighbouring objects. A bisector between two line segments or two points is a straight edge (Figure 3.12a and Figure 3.12d), and between a point and a segment is a parabola (Figure 3.12c). A bisector for an endpoint of a decomposed segment and its interior is a straight line perpendicular to the segment passing its endpoint location (Figure 3.12b), and it can be called a separator (Imai (1996)). So in contrary to the ordinary VD, the cells in the LSVD are not necessarily convex, as the shape of Voronoi edges between points and line segments is parabolic.

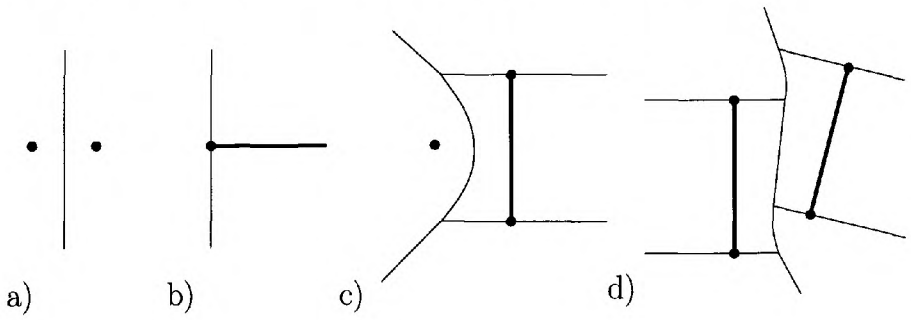


Figure 3.12: Bisectors between two objects in the LSVD, where objects are a) two points; b) an endpoint and a segment; c) a point and a segment; d) two segments.

The dual graph of the LSVD forms a planar Voronoi Adjacency Graph (VAG) which expresses adjacency relationships of point and line segment objects of the LSVD (Gold (1990)). It consists of arbitrarily shaped edges (straight lines or curves) connecting neighbouring points and line segments, as shown in Figure 3.13a. When an edge originates at a line segment it is drawn from its middle point (Yang (1997)). The VAG for simplicity is usually drawn using straight lines, as shown in Figure 3.13b. The VAG, although it consists of triples that are not straight edge triangles, is often called a Delaunay triangulation as for each triple of objects there exists a circumcircle tangent to them that is empty of other objects (its circumcentre is a Voronoi vertex). Figure 3.13a shows a circumcircle of a triple consisting of two line segments a and b and a point c. It is centered at a Voronoi vertex v, is tangent to a and b and goes through c.

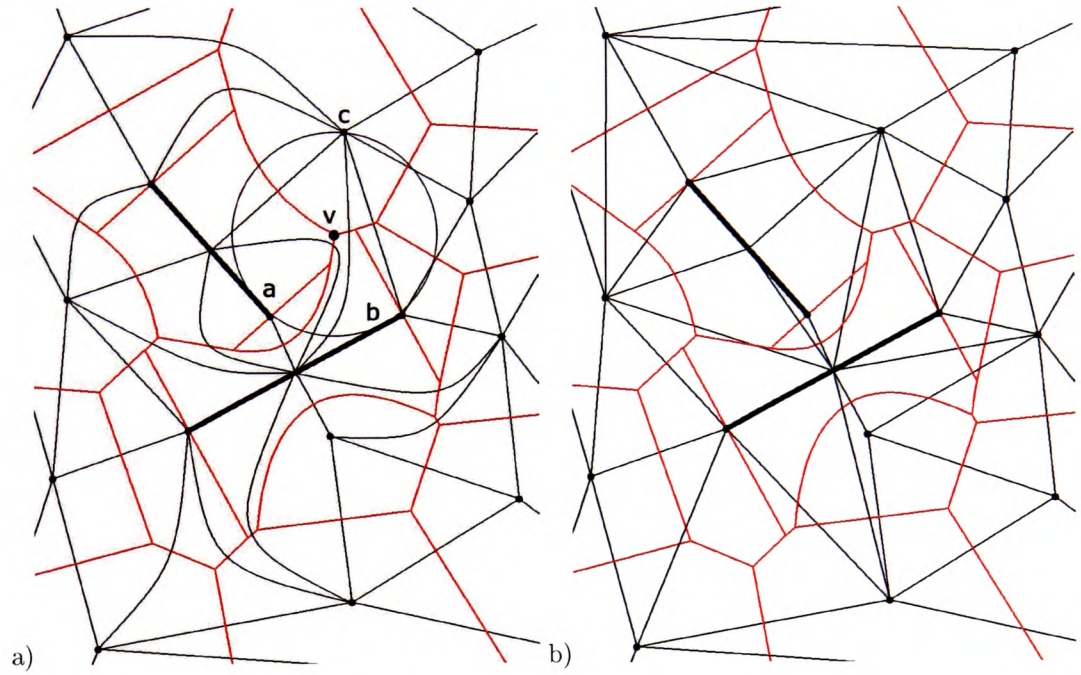


Figure 3.13: The LSVD and it dual graph VAG. a) VAG drawn using arbitrarily shaped edges. b) VAG drawn using straight lines.

3.8 Crust/Skeleton

The Voronoi diagram and Delaunay triangulation are linked with two other geometric structures, known as the "crust" and the "skeleton". The skeleton is associated with the well known "Medial Axis" term, introduced by Blum (1967) who used it for solving biological problems. He showed that the medial axis for polygons (see Figure 3.14) consists of points equidistant to at least two points on the polygon boundary.

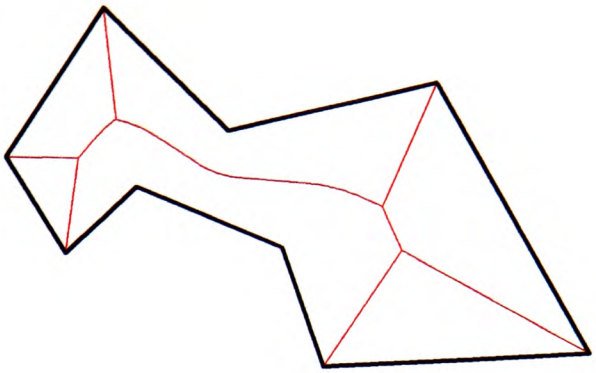


Figure 3.14: Medial axis (thin lines) of a polygon (thick lines).

The term "crust" was introduced by Amenta et al. (1997) who attempted to reconstruct the shape of the curve from the Delaunay triangulation of an unordered set of samples of that curve. A set of nodes V was triangulated and the Voronoi diagram was extracted (Figure 3.15a). The nodes of the resulting VD were added to the triangulation and the edges of the resulting triangulation connecting nodes of V formed the crust (black edges in Figure 3.15b). They also provided an alternative definition, showing that for a triangulated set of nodes V and its VD the crust is formed by edges connecting vertices of V for which there is a circle empty of Voronoi vertices that passes through the endpoints. It can be noticed that the crust is a subset of the Delaunay triangulation. The graph approximating the medial axis ("straight line medial axis") and consisting of dual Voronoi edges (thick edges in Figure 3.15c) of Delaunay edges (thin dashed edges in (thick edges in Figure 3.15c) not belonging to the crust (thin solid edges in Figure 3.15c) is called in their work the anti-crust, while other researchers often use the term "skeleton".

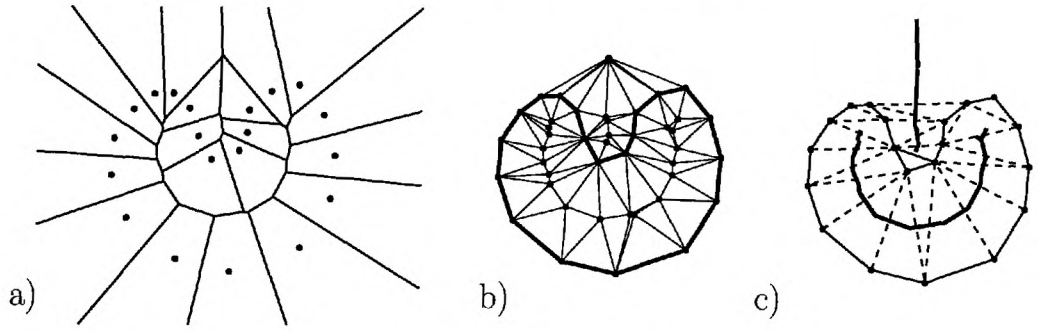


Figure 3.15: Amenta's crust extraction algorithm's result. a) The Voronoi diagram of samples of a curve. b) The crust (thick edges) extracted after triangulating the samples and Voronoi nodes. (after Amenta et al. (1997)) c) The skeleton (thick edges) being a subset of the Voronoi diagram.

Gold (1999) and Gold and Snoeyink (2001) simplified Amenta's algorithm for the extraction of the crust. They shown that, in every Delaunay/Voronoi edge pair, either the Delaunay edge could be assigned to the crust or else the dual Voronoi edge could be assigned to the skeleton. The Delaunay edge belongs to the crust when there exists a circle through its two vertices that does not contain either of its associated Voronoi vertices; otherwise the corresponding Voronoi edge belongs to the skeleton. In practice their approach means applying the empty circumcircle test for each Delaunay/Voronoi edge pair.

Figure 3.16a shows a Delaunay edge connecting points *p1* and *p2* and its dual Voronoi edge having *v1* and *v2* as it vertices. The circle *p1,v1,p2* does not contain the second Voronoi vertex *v2* inside, so the Delaunay edge belongs to the crust. Figure 3.16b shows a case, when the *v2* vertex lies inside the circle, so the Voronoi edge belongs to the skeleton.

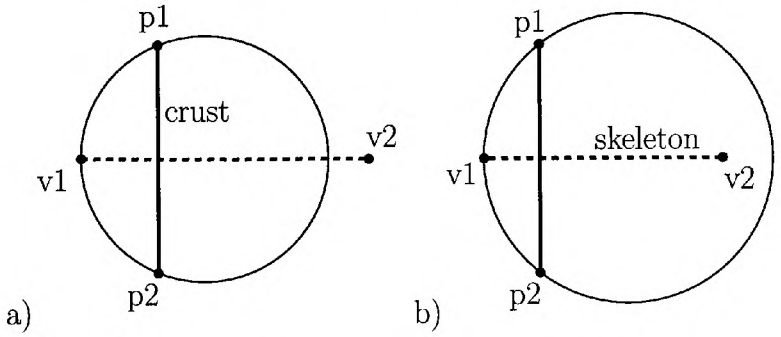


Figure 3.16: Results of the crust/skeleton test. a) The Delaunay edge p1-p2 is a part of the crust. b) The Voronoi edge v1-v2 is a part of the skeleton.

In the case of the LSVD of points and line segments the skeleton can be extracted also, using the same test for edges connecting pairs of points and additionally assigning each edge having a line segment as its endpoint to the skeleton, as shown in Figure 3.17a. It is also possible to extract "the central skeleton" only, by rejecting skeleton edges for edges having two connected line segments as vertices (Figure 3.17b).

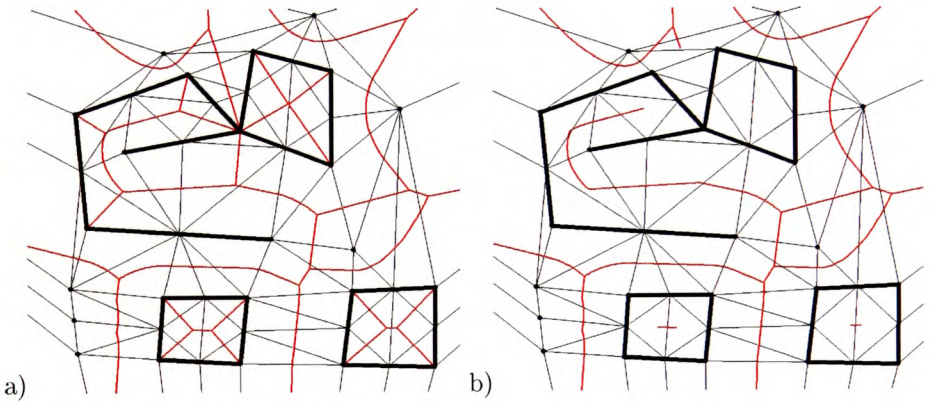


Figure 3.17: The skeleton in a LSVD. a) The complete skeleton. b) The central skeleton only.

It is also worth mentioning another type of skeleton - a "labelled skeleton", used by Gold et al. (1996) for digitizing polygon maps. For each polygon they generated fringe points near its boundary (with different labels for each polygon) and computed the Voronoi diagram of all fringe points. The labelled skeleton consists of Voronoi edges separating points having different label values, or in other words it consists of dual edges of Delaunay edges connecting points with different labels (see Section 6.2.3 for examples). Labelled point skeletons are useful where geometric positions are not sufficient to extract meaningful geometric skeletons.

Chapter 4

Previous Voronoi and Delaunay Algorithms

This chapter presents existing algorithms used to create, manage and modify Voronoi diagrams, Delaunay triangulation and other diagrams. The point based VD/DT construction algorithms fall into two main categories - batch and incremental. "Divide and conquer" and sweep line algorithms are the two most popular batch methods. They create the triangulation in one operation using the whole set of vertices as the input and insertion of an additional node requires repeating the whole mesh construction process again. Incremental methods build the triangulation by inserting nodes incrementally one by one. Linear features can be incorporated into VD/DT diagrams as constrained Delaunay edges, conforming Delaunay edges or line segment objects having their own Voronoi regions and various methods of adding and removing linear features are presented in this chapter as well. Aurenhammer (1991) provides a good description of the evolution of the triangulation construction methods.

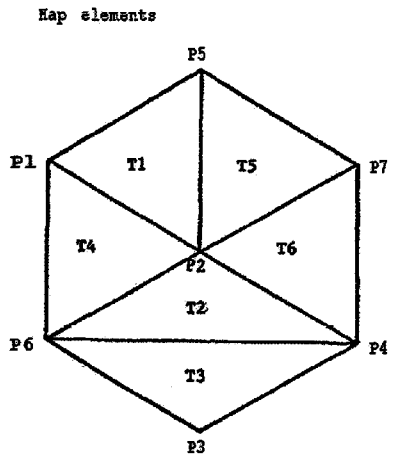
However, first of all the VD/DT diagrams need a data structure that allows their construction, storage and update so the chapter starts with a description of various data structures. These include the popular triangular element structure and various types of edge-based structures, including the quad-edge structure used in this work.

4.1 Quad-Edge and Other Data Structures

Polygonal planar partitions consist of vertices, edges and polygons, all connected and related to each other. Based on the topology of the subdivision a data structure defines how each element is stored and what references to its neighbourhood are needed to be able to navigate between elements in the mesh (Botsch et al. (2002)). Face-based and edge-based structures are the two main groups of data structures. Face-based data structures use a face as a record with pointers to its vertices and neighbouring faces. Edge-based structures use an edge as a record with pointers to its endpoints and pointers to neighbouring edges or faces. Fixed numbers of pointers for edges make storing meshes with variable faces easier with edge-based structures than face-based ones.

The triangular element structure is the traditional choice for implementing triangulations and has been known and used for more than 30 years (Gold (1976); Lawson (1977); Lee and Schachter (1980)). Its popularity is due to its simplicity and the fact that storing the triangulation as a set

of separate triangles seems to be the first choice for most people. Each triangle of the mesh is stored as an object with six pointers - to the three associated vertices of the triangle and to the three adjacent triangles, as in Figure 4.1. Edges are not stored explicitly, but can be extracted if necessary.



Data elements

47

Triangles	Adjacent Triangles	Associated Data Points
1	5,_,4	1,2,5
2	3,6,4	2,6,4
3	2,_,_	3,4,6
4	2,1,_	1,6,2
5	1,6,_	7,5,2
6	2,_,5	7,2,4

Points

Point #	X	Y	Z (elevation)
1	X_1	Y_1	Z_1
2	X_2	Y_2	Z_2
3	X_3	Y_3	Z_3
4	X_4	Y_4	Z_4
5	X_5	Y_5	Z_5
6	X_6	Y_6	Z_6
7	X_7	Y_7	Z_7

Figure 4.1: Triangular element data structure. (from Gold (1976))

There are many different edge-based structures and often many variations of them. The oldest probably is the winged-edge structure introduced by Baumgart (1975) where each edge of the mesh is saved as a record. It stores pointers to two endpoints of the edge, pointers to the left and right faces, pointers to two edges adjacent to the left side of the edge and pointers to two edges adjacent to the right side. Another popular structure is the half-edge (Mantyla (1988)), also known as doubly connected edge list (DCEL), where each edge of the mesh is stored as a pair of oppositely oriented edges - half-edges, so two records are used to represent one edge of the mesh. Each half-edge record contains a pointer to the vertex of the mesh being the endpoint of the half-edge, a pointer to the adjacent oppositely oriented half-edge, a pointer to the incident face (not needed when representing a graph) and a pointer to another half-edge around the face.

The quad-edge is the third popular structure. Since it has been used in this work it will be described in detail. The structure was designed by Guibas and Stolfi (1985) to store edges of a planar subdivision and its dual graph simultaneously, and allow navigation within both graphs easily (which is very useful for working with Voronoi diagrams and Delaunay triangulations). The structure is popular because it is elegant and easy to use, allows simultaneous access to both dual graphs and the authors provided a detailed description of its usage.

Each edge of the subdivision is represented by a group of four linked half-edges (quads) - two for the primal edge and two for the dual edge, as in Figure 4.2a. Each of them contains three

references: *Org* to its endpoint (an object of the TPoint class managing two dimensional points), *Rot* to its dual half-edge in the group and *Onext* ("Origin Next") to the next counterclockwise half-edge in the graph (first left).

```
TQuad = record
  Org:TPoint;
  Rot: TQuad;
  Onext: TQuad;
End;
```

A group of four linked together half-edges representing an edge of the graph is called a "quad-edge" and is created using a MakeEdge operation, described later in this section. Figure 4.2b shows a frequently used in diagrams symbolic cross representation of an edge connecting points a and b represented by the quad-edge structure. The disk mark inside the cross marks the current half-edge being processed (here the quad e having the point a as the origin).

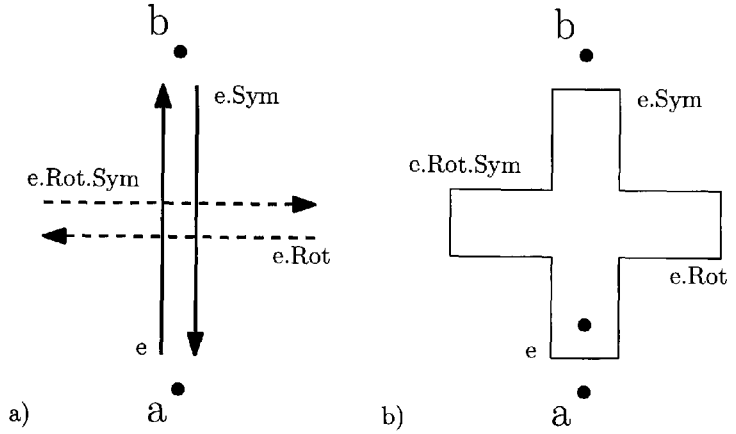


Figure 4.2: An edge connecting points a and b represented by the quad-edge structure. a) Four linked half-edges. b) A symbolic cross representation.

There are some additional higher level operators simplifying use of the structure. The most used are *Sym* returning the half-edge with the opposite direction, *Dest* returning the destination vertex of the edge and *Oprev* returning the next clockwise edge.

```
e.Sym = e.Rot.Rot
e.Dest = e.Sym.Org
e.Oprev = e.Rot.Onext.Rot
```

Figure 4.3 shows a configuration of five edges represented by quad-edges. Various quads are referenced from the quad e of the quad-edge joining points a and b using different operators . Also, calling the *Rot* operator four times for e returns the same edge e (e.Rot.Rot.Rot.Rot = e).

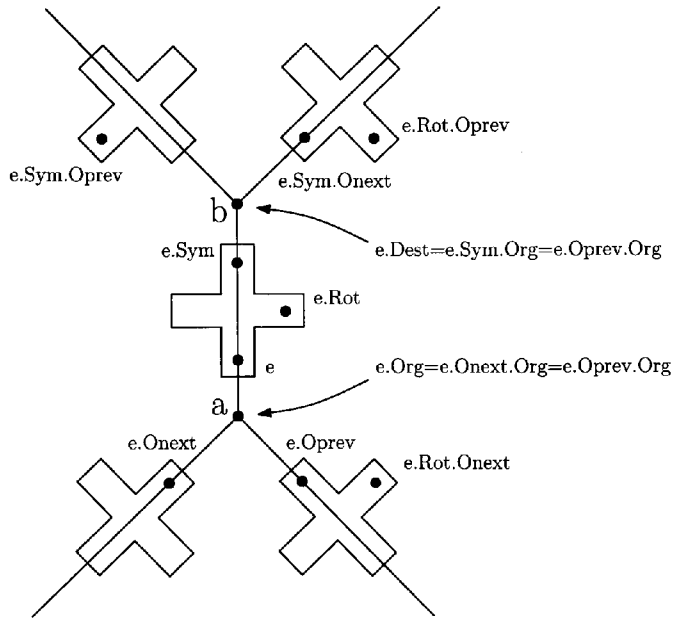


Figure 4.3: Five edges of a graph represented by quad-edges and various quads referenced from the quad e.

The main advantage of representing graphs using quad-edge is a simultaneous direct access to both the primal and dual graphs. Navigation using the operators is simple and intuitive. The construction of the graph requires only two operators to make edges and join them together. The main disadvantage of the quad-edge structure is its high storage cost as for for each edge it uses 3*4=12 references (3 references in each of its four quads).

One of the main advantages of the quad-edge structure is that a graph can be constructed using only two basic topological operators: MakeEdge and Splice. MakeEdge creates a new edge linking two existing nodes of the graph. Splice joins edges together or disconnects them.

Function MakeEdge creates four new half-edges and initializes them - linking them together and pointing the origins of two of them to the vertices of the graph. The origins of the two other half-edges are not set initially and Voronoi vertices are later assigned to them. The function returns the pointer to the first of the new quads.

```
function MakeEdge(Org, Dest : TPoint) : TQuad;
var
  Q0, Q1, Q2, Q3 : TQuad;
begin
  //create four new 1/4 quads
  Q0 := TQuad.Create; Q1 := TQuad.Create;
  Q2 := TQuad.Create; Q3 := TQuad.Create;
  //link the four parts
  Q0.Rot := Q1; Q1.Rot := Q2;
  Q2.Rot := Q3; Q3.Rot := Q0;
  //link 0 & 2 to themselves, 1 & 3 to each other
  Q0.Onext := Q0; Q1.Onext := Q3;
  Q2.Onext := Q2; Q3.Onext := Q1;
  //link 0 & 2 to themselves, 1 & 3 to each other
  Q0.Org := Org; Q2.Org := Dest;
End ;
```

Figure 4.4 shows the result of the function. A new quad-edge is created between the *Org* and *Dest* points. *Rot* pointers are set from each quad to the next counterclockwise quad. Quads *q0* and *q2* are set with *Onext* pointers referencing themselves, while *q1* and *q3* point at each other. *Org* pointers are set as well, with *q0.Org* pointing at *Org* and *q2* pointing at *Dest*.

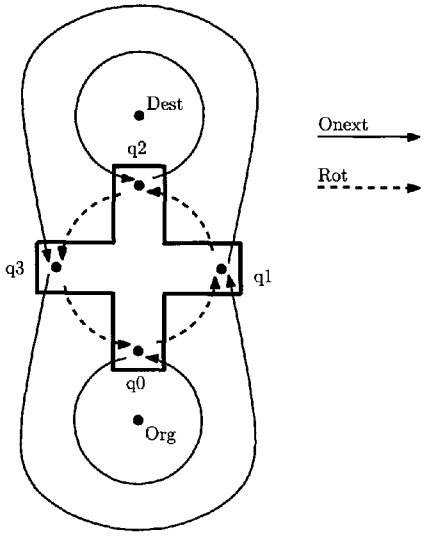


Figure 4.4: The result of MakeEdge function.

Operator Splice is used to connect or disconnect edges. It is its own inverse – performing the same operation twice results in the initial state of the graph. It takes two edges as parameters and affects rings of edges around their origins. Two parameter edges determine where the rings will be cut or merged. If two rings are distinct Splice combines them into one. If edges are a part of the same ring the operator disconnects them.

```
procedure TQuad.Splice(a,b : TQuad);
var
  anr, bnr, an, bn, anrn, bnrn : TQuad;
begin
  an := a.Onext;        bn := b.Onext;
  anr := an.Rot;        bnr := bn.Rot;
  anrn := anr.Onext;    bnrn := bnr.Onext;
  a.Onext := Bn;        b.Onext := an;
  anr.Onext := bnrn;    bnr.Onext := anrn;
end;
```

Figure 4.5 shows the Splice operation for two configurations of edges. Edges used in the process are marked according to the listing above and the new connections are marked with arrows. Figure 4.5a shows the process of disconnecting two edges a and b, leading to a separation of edge a from b and its adjacent edge. Figure 4.5b shows another usage of Splice for connecting two loops of edges represented by a and b. Here we can see that the selection of a and b edges passed to Splice as arguments is crucial - using other edges would have lead to a different result.

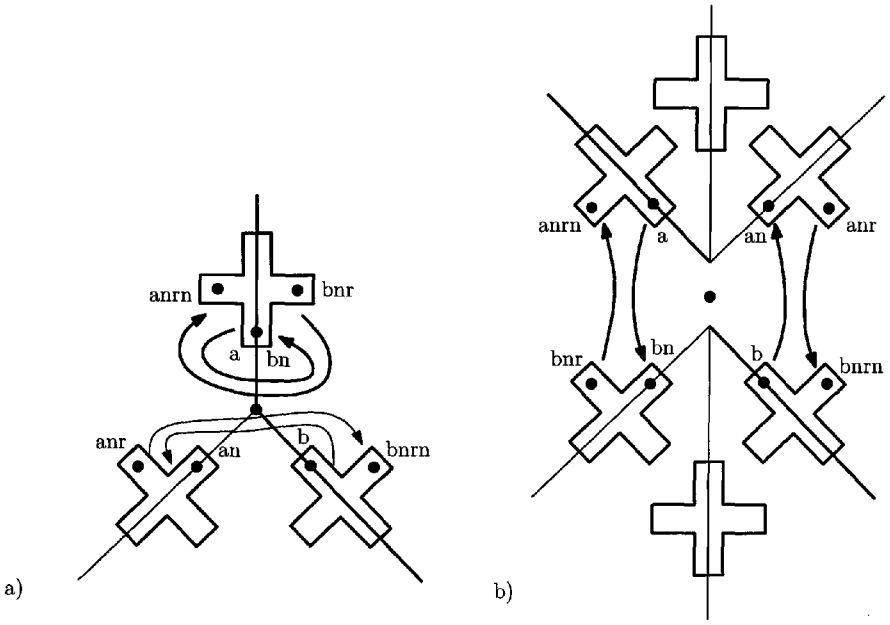


Figure 4.5: The result of Splice operations with new Onext connections marked. a) Disconnecting edges a and b. b) Connecting edges a and b.

4.2 Batch Methods

Batch methods are the fastest methods of triangulating data (although more difficult to implement than incremental algorithms). They are used to construct the triangulation of the whole set at once, so all points have to be known before the process. An addition of a new point requires construction of the whole triangulation from the beginning or the incremental approach. The two main batch techniques are divide and conquer and sweepline.

The divide and conquer algorithm for construction of Delaunay triangulations was proposed by Lee and Schachter (1980) and was presented for the quad-edge structure in detail together with its proof by Guibas and Stolfi (1985). It is based on the idea of dividing the data into pieces, finding Delaunay triangulations for them and merging. The process starts with splitting the input data points into two left and right halves according to their x coordinates. Then the halves are recursively triangulated and merged. The algorithm computes the DT of n sites in $O(n\log n)$ time.

The sweepline algorithm was proposed by Fortune (1987) and is based on the idea of moving a conceptual horizontal line upwards through the data set and triangulating the data accordingly. It is competitive in its simplicity with incremental algorithms, runs in $O(n\log n)$ time and is much easier to implement than the divide and conquer method. However, it has been shown to be sensitive to near-degenerate cases.

4.3 The Incremental Algorithm

The incremental algorithm allows creation of the VD/DT by inserting new points into the diagram incrementally, one by one. It is based on the empty circumcircle and the orientation geometric tests. Nodes of the diagram can be located using a simple "Walk" operation and the existing vertices can be removed in an incremental manner as well. All these operations are presented in

this section.

4.3.1 Initialization – The Big Triangle

Most incremental triangulation algorithms assume that points have to be inserted inside a specified very large convex polygon, usually a triangle (Gold and Maydell (1978); Guibas and Stolfi (1985)). The size of the polygon is specified before constructing the mesh. Usually it is possible to determine the range of the data before construction the mesh (extracting the minimal and maximal coordinates) and use those values to set the size of the bounding triangle. The triangle is created by creating three new edges and linking them together using Splice operators. Then data points can be inserted inside, as in Figure 4.6. The advantage of this approach is that it simplifies greatly all operations, since we are sure that an inserted point is inside the polygon and we do not have to deal with complex issues of adding vertices outside the convex hull. Two main disadvantages are that the data has to be processed first to determine the size of the bounding triangle and additional triangles are created connecting the data points with the big triangle.

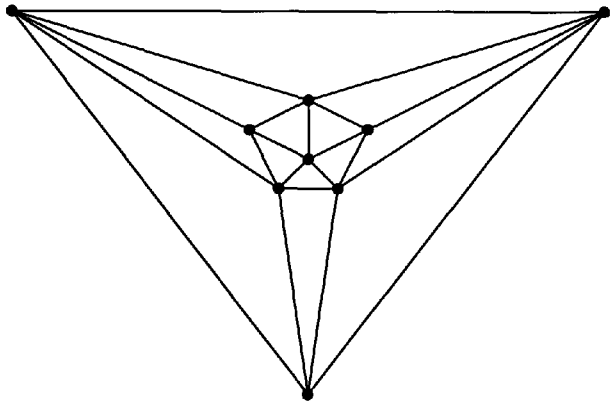


Figure 4.6: The big triangle and a simple triangulation.

4.3.2 Predicates

Delaunay Triangulation construction and manipulation requires only two geometric tests – the orientation of three points (CCW) and the empty circumcircle property (InCircle) (Guibas and Stolfi (1985)).

$$CCW(a,b,c) = \begin{vmatrix} x_a & y_a & 1 \\ x_b & y_b & 1 \\ x_c & y_c & 1 \end{vmatrix} \tag{4.1}$$

The sign of the determinant of three points CCW provides information about their clockwise (cw) or counterclockwise (ccw) orientation, as shown in Figure 4.7. If three points are collinear the determinant equals zero. The test is used in many processes, including the point location and mesh traversal. CCW can be also used to test on which side of an oriented segment ab lies another point c. The point c is on the left side of ab if the orientation of three points a, b and c is counterclockwise (Figure 4.7a), otherwise the point c is on the right side of ab (Figure 4.7b).

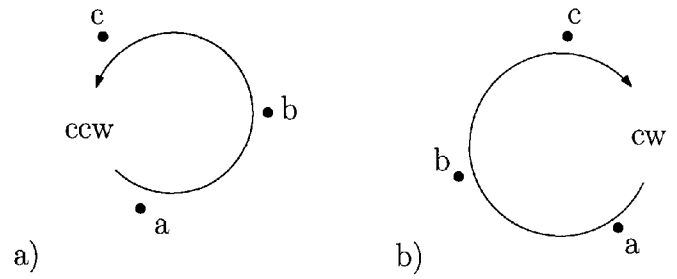


Figure 4.7: The Orientation of three points. a) Counterclockwise. b) Clockwise.

$$InCircle(a,b,c,d) = \begin{vmatrix} x_a & y_a & x_a^2+y_a^2 & 1 \\ x_b & y_b & x_b^2+y_b^2 & 1 \\ x_c & y_c & x_c^2+y_c^2 & 1 \\ x_d & y_d & x_d^2+y_d^2 & 1 \end{vmatrix} \tag{4.2}$$

The operator InCircle tests if a point lies inside a circle defined by three points. It returns a positive value if the point is inside, negative if outside and zero if it lies exactly on the circle. The test is the key element of the DT point insertion, where the empty circumcircle property of the DT is tested after addition of a new point.

Changing the order of arguments passed to CCW and InCircle tests when using floating-point arithmetic may give different results. To assure the uniformity of their results they should be called with arguments in the same order, for example by reordering them according to their coordinates, so the point with the "smallest" coordinates is always passed as the first argument.

4.3.3 Point Location Methods

A point location is an important component of the mesh construction process and can be described as locating a triangle containing a query point p with x and y coordinates. It is an operation performed in point insertion and deletion, interpolation and many other operations.

The fastest existing methods are usually very complex, often assume that the subdivision is fixed and require additional storage or data structures that additionally have to be pre-processed. In practical applications sub-optimal methods, simple in implementation and still fast, are usually preferred (Devillers et al. (2002)). Most of them use relationships between the triangulation simplices for the navigation and need no additional structures. There are several different walking strategies, including the straight, orthogonal, visibility and stochastic walk (Devillers et al. (2002)). The visibility walk (Green and Sibson (1978); Guibas and Stolfi (1985); Gold et al. (1977)) is the most popular strategy for Delaunay triangulations. The idea of the method is to start from an arbitrary location and move iteratively in the general direction of point p until a triangle enclosing p is found, as in Figure 4.8.

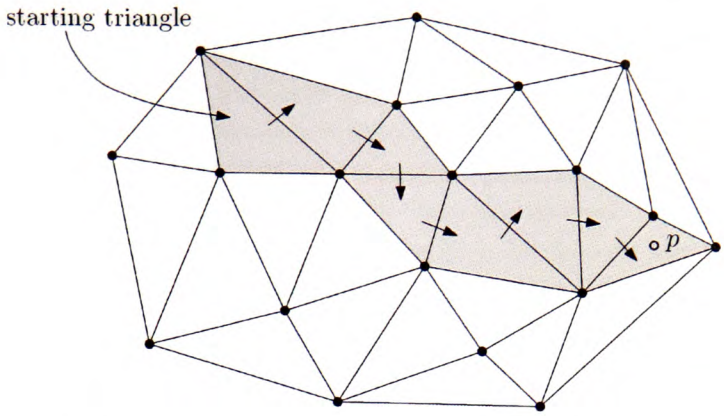


Figure 4.8: Walk in the DT. (from Ledoux (2006))

The method requires two tests only - one to determine which of two neighbouring triangles (initially three) to walk to and another to test if the point is enclosed by a triangle. The selection of the next triangle means applying the orientation test - the adjacent triangle is used if the common edge e separates the current triangle and point p (the sign of *CCW(e.Org, e.Dest, p)* is tested). If both adjacent triangles pass that test then any of them is selected randomly.

Setting the starting location, which can be a triangle or edge depending on the data structure used, is a crucial part of the walking process, greatly influencing the speed of the search. Most of the practical implementations reset the starting location after inserting or deleting a vertex by setting it to one of the recently updated edges or triangles. In practical applications like recreating terrain models from samples, consecutive data points inserted into the triangulation tend to be close to each other and the walk algorithm requires very few operations to locate the appropriate triangle for the insertion.

4.3.4 Vertex Insertion in the Delaunay Triangulation

Incremental algorithms create a mesh by inserting vertices one by one and Delaunay properties are preserved after adding each node. One important property of incremental algorithms is that the insertion of a new node requires a modification of the neighbourhood of that node only, the rest of the mesh remains unchanged, so it is the mandatory method for constructing meshes for dynamic and kinetic applications. Incremental algorithms are usually slower than batch methods, mostly due to a triangle location operation needed before the actual insertion. Using special walking techniques, also with additional structures, may significantly speed up the insertion process.

There are two main ways of inserting a node into the triangulation - by flipping edges (switching the diagonal of a quadrilateral) in the vicinity of a new node to preserve Delaunay properties (Gold et al. (1977); Guibas and Stolfi (1985) or by making a hole defined by the neighbours of the new node and re-triangulating it (Bowyer (1981); Watson (1981)).

Lawson (1972) proved that any triangulation, including a Delaunay triangulation, can be created from an arbitrary triangulation by flipping edges. Gold et al. (1977) used the flipping method for the arbitrary triangulation optimization after inserting a new node into an existing triangle and discussed various geometric criteria deciding about the flip. Lawson (1977) used a similar method for Delaunay triangulations, inserting a new node by splitting an existing triangle into three (unless the node is located exactly on one of the edges) and provided three different criteria for choosing a

preferred triangulation of a convex quadrilateral: the Min-Max angle criterion, the circle criterion and the Thiessen region criterion. Lee and Schachter (1980) applied the Min-Max angle criterion in their incremental insertion. Guibas and Stolfi (1985) provided the most detailed description of the incremental insertion of nodes for Delaunay triangulation. They used the flipping method (swap) together with the empty circumcircle Delaunay criterion. Their algorithm starts with location of the triangle containing the location of the new node. Then the new node is inserted by splitting the triangle into three new triangles, as in Figure 4.9 (top left). In the next step the triangulation in the vicinity of the new node is optimized until it is Delaunay (Figure 4.9). The three new triangles are tested using the InCircle function if they satisfy the empty circumcircle property. If the outer vertex of the triangle adjacent to the tested triangle T lies inside the circumcircle of T then their common edge is flipped (Figure 4.9 (top second)), creating two new triangles whose circumcircles will be tested as well. Such tests are carried out until all the triangles adjacent to the new node are Delaunay.

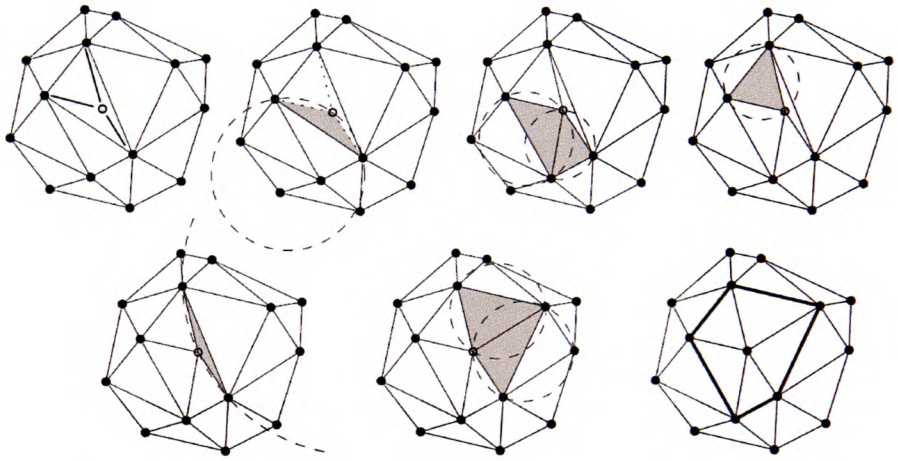


Figure 4.9: Incremental insertion of a vertex (from Ledoux (2006)).

The Bowyer-Watson algorithm was developed independently by Bowyer (1981) and Watson (1981) and published in the same issue of of the same journal. It creates a Delaunay triangulation by incrementally adding new points and modifying the existing triangulation by means of a sequence of local operations. In this method all triangles whose circumcircles contain the new node are deleted (Figure 4.10a), creating a hole in the triangulation. In the next step the empty area is retriangulated by joining the new vertex with all the vertices of the hole (Figure 4.10b) which produces a Delaunay triangulation, as no other triangle contains the new point inside.

4.3.5 Vertex Deletion in the Delaunay Triangulation

Deletion is the inverse operation to insertion. It has been proved that a vertex v having degree k can be deleted from a Voronoi diagram in $0(k)$ time (Chew (1986); Aggarwal et al. (1989)). The sub-optimal solutions are usually preferred due to their simplicity. Some of them are based on the idea of making a hole by removing the vertex and adjacent edges and retriangulating the resulting polygon (Bossen (1996)). Other algorithms use the idea of flipping edges adjacent to the deleted vertex in a certain order.

Among the flipping methods the ones based on the concept of ears of a polygon are the most

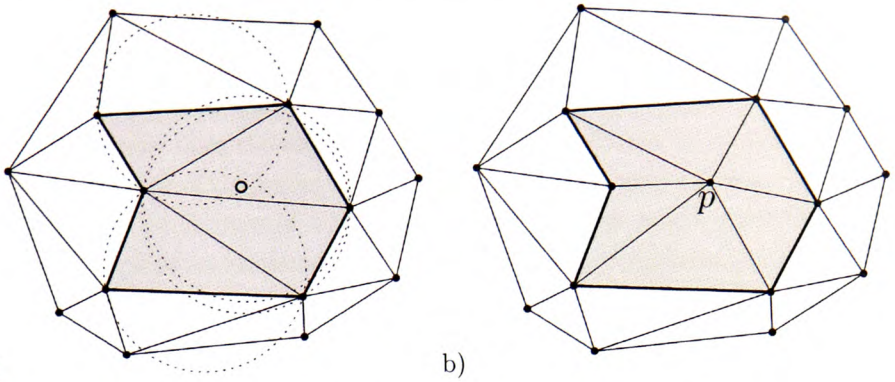


Figure 4.10: Bowyer-Watson insertion algorithm. a) Triangles containing the new point location. b) The resulting triangulation after inserting a new point p. (from Ledoux (2006))

used. The idea was proposed by Heller (1990) and modified by Devillers (1998). The complexity is $0(k\log k)$, where k is the number of neighbours of the deleted node p, forming a star-shaped polygon H (Figure 4.11). An ear (also called an "imaginary" triangle) is formed by three consecutive vertices v_1,v_2 and v_3 of H if the line segment connecting v_1 and v_3 lies inside the polygon H and does not cross its boundary. In practice determining if three points form an ear means applying the CCW test. Both algorithms "fill the hole" of H adding a new ear to the triangulation (cutting off an ear from the polygon) one by one until p is connected to three neighbours only. Then the remaining three edges are removed together with the vertex. Adding an ear v_1,v_2, v_3 is achieved by flipping the edge connecting the deleted vertex p and v_2, which makes v_2 no longer connected to p. The order of adding ears is the critical part of both algorithms. Heller proposed adding ears according to their circumcircle size, claiming that the ear with the smallest circumcircle radius is Delaunay. Devillers proved that Heller's test fails in certain cases and proposed adding ears in the order of increasing Power values instead, computed using the InCircle and CCW tests:

$$Power(v, v_1, v_2, v_3) = \frac{InCircle(v, v_1, v_2, v_3)}{CCW(v_1, v_2, v_3)} \tag{4.3}$$

Figure 4.11a shows a triangulation set with a point p before its deletion, being connected to six other points. Figure 4.11b shows the result without the point p after swapping appropriate edges and removing the last three edges connecting p with the vertices of the triangle enclosing p.

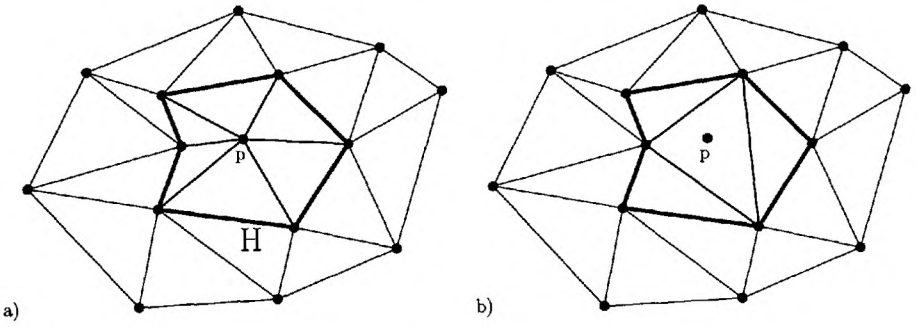


Figure 4.11: A vertex deletion. a) The point p to be deleted. b) The resulting diagram after flipping three edges, removing the three remaining ones and deleting p. (after Devillers (1998))

Mostafavi et al. (2003) proposed another ear based solution, simpler in implementation, but

less efficient, with $O(k^2)$ complexity. Unlike the Devillers' method, requiring a priority queue for managing the order of operations, Mostafavi's solution does not need any additional structures. In his method the circumcircle of each ear is tested against the remaining vertices of the "hole" and if all of them fall outside then the ear is added to the triangulation by switching an appropriate edge. For a small number of neighbours Mostafavi's method is comparable in efficiency and since the average number of neighbours is six, it can be used in applications where the distribution of nodes is regular. It's efficiency decreases with the increase of the neighbours, so if the data is highly anisotropic in distribution like bathymetry data, then using Devillers' method is advised. Another disadvantage is the need of a tolerance value for solving cases with four or more cocircular points.

4.4 Mesh Traversal Methods

Traversal of the mesh means visiting each of its components (edges, nodes or cells) and reporting them once only. It is an operation used for drawing the mesh or processing all its components. This can be achieved using traditional graph traversal methods and breadth first search (BFS) and depth first search (DFS) are probably the simplest and most popular methods (Sedgewick (1992)).

In breadth first search the algorithm begins from a selected node, marks it as visited (by using a mark attribute or saving it in an additional queue, list or stack) and explores and stores in a chosen data structure all its neighbours. Nodes are manipulated in FIFO order (First In, First Out). After visiting all neighbours each of the stored nodes is processed, by marking it as visited and this time its neighbours are explored. If any of them has not been explored yet then it is stored for processing at the end of the storage structure. The process ends when all the nodes are marked as visited and there are no nodes left for processing in the data structure. This method with slight modifications can be used for visiting all edges or faces of the graph.

In depth first search extremes are searched first, so the process goes as far from the starting node as possible, unlike BFS where near neighbours are prioritized. DFS starts from a selected node, marks it as visited and proceeds to the first neighbour. The neighbour is marked as visited and the algorithm proceeds to the first neighbour of that node, going further away from the starting point. The process is repeated until one of the already visited nodes is approached. In such a case another neighbour is explored and if it has not been visited the process continues from that neighbour further. If all neighbours are marked as visited then the process backtracks to the previous node and checks its neighbours.

These methods require using additional flags to mark visited components and/or additional data structures for processing them, which increases the storage of the mesh. Also before each traversal or search, marks have to be reset, which increases the overall time of the operation (unfeasible for very large data sets).

If the graph is a Delaunay triangulation, then it can be traversed using an old, simple and fast method called Scan, that does not require any additional marks or structures. It was proposed by (Gold and Maydell (1978); Gold et al. (1996)) and is based on the observation, that any DT in 2D can be ordered as a binary tree from a fixed viewpoint (Ledoux (2006)), so every entered triangle has two triangles as children. It allows visiting all nodes, edges or triangles of the triangulation and only requires a specified viewpoint for the operation and a stack for edge processing.

The method starts from an arbitrary triangle, usually the nearest to the viewpoint. Edges of the current triangle are numbered in counterclockwise order. The edge number 1 is the edge from which the triangle was entered from the parent triangle and edges number 2 and 3 (in the ccw

order) are stacked to be considered for the next move. Then an edge is popped from the stack and the adjacent outer triangle is classified (using the viewpoint) to one of three categories: 2U3U, 2U3D and 2D3U, as in Figure 4.12. U and D ("up" and "down") state if the edge is facing towards or outwards from the viewpoint and is computed using a simple determinant of the endpoints and the viewpoint. 2 and 3 are the numbers of the edges of the triangle, so for example 2U3D means that in this triangle the edge 2 faces up and edge 3 faces down. If the triangle is 2U3U then both its neighbouring triangles are stacked. If it is 2U3D only the triangle adjacent to the third edge is stacked, and in case of 2D3U none of the adjacent triangles are stacked. Using this classification all components of the mesh are visited once only.

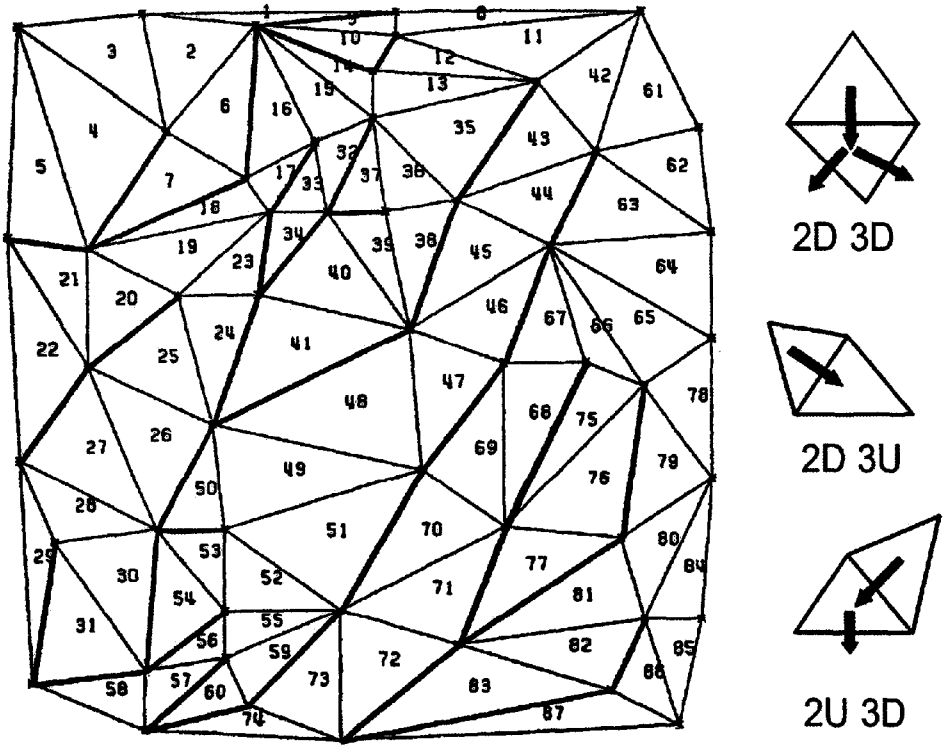


Figure 4.12: The DT traversal - the traversal order (left) and the three kinds of triangles (right, from Gold et al. (1996)).

4.5 Methods of Moving Points in the VD/DT

There are various methods of maintaining a point motion in the Voronoi Diagram and Delaunay Triangulation. One of the first was developed by Roos (1993) (further investigated by Albers et al. (1998)) and is based on processing topological events with swap operations. A set of nodes V moving with a constant velocity along specified trajectories is considered. In the first step the Delaunay triangulation of V is computed. Then all existing quadrilaterals (pairs of adjacent triangles) are processed, all potential topological events are computed and put in a priority queue according to the time their arise. In the next step the first topological event is extracted from the queue, the position of the appropriate point is updated and the diagonal of that quadrilateral is swapped. A swap operation affects four quadrilaterals, which consist of both triangles adjacent to the swapping edge and the outer triangles adjacent to those two triangles. After each swap four

elements in the priority queue corresponding to changed quadrilaterals are updated as well. The process terminates when the priority queue is empty.

A different approach was proposed by Gold (1990), Gold (1991) and Gold et al. (1997). He uses a model with a single "moving point" *MP*, moving along a specified trajectory among a fixed set of generators defining the Voronoi Diagram and connected by the edges of the underlying DT. His method is also based on processing topological events. Each triangle of the DT has an associated circumcircle and the trajectory of the movement usually crosses or touches a number of circumcircles, depending on the length of the trajectory and complexity of the model. A circumcircle crosses the trajectory once only if the endpoint of the trajectory lies inside the circumcircle or twice when both endpoints of the trajectory are outside the circumcircle. The places where circumcircles contact the trajectory specify the locations of topological events. The events are processed consecutively from the nearest to the moving point to the furthest, and the movement of the point occurs as a series of discrete steps.

The *MP* movement starts with finding the first topological event location by testing circumcircles of triangles surrounding *MP*. As *MP* moves it enters or leaves circles, losing or gaining a neighbour.

There are two kinds of circles that are processed. One group are circumcircles of triangles external to triangles adjacent to *MP*, as in Figure 4.13a, with two vertices a and c connected to *MP* and b separated from *MP* by edge ac (as marked in Figure 4.14a). Those triangles will be called the real triangles, and their circumcircles - the real circles.

The other group are circumcircles of triangles formed by each counterclockwise oriented triple a, b, c of consecutive points connected to *MP*, (as marked in Figure 4.15a). Such triples form the imaginary triangles (as shown in Figure 4.13b), and their circumcircles will be called the imaginary circles. Imaginary circles do not exist in the DT, but could be created during the *MP* movement by flipping edges to disconnect neighbours from the moving point. *MP* can only enter the real circles and leave the imaginary ones, by moving onto their boundary from outside or inside.

If we consider the configuration of circles from Figure 4.13 then the closest event is caused by one of the real circles, so *MP* is moved to the intersection point and the appropriate edge is flipped, which adds a new neighbour to *MP*. Then circles are calculated for the new DT configuration and the whole process is repeated until there are no topological events found on the trajectory.

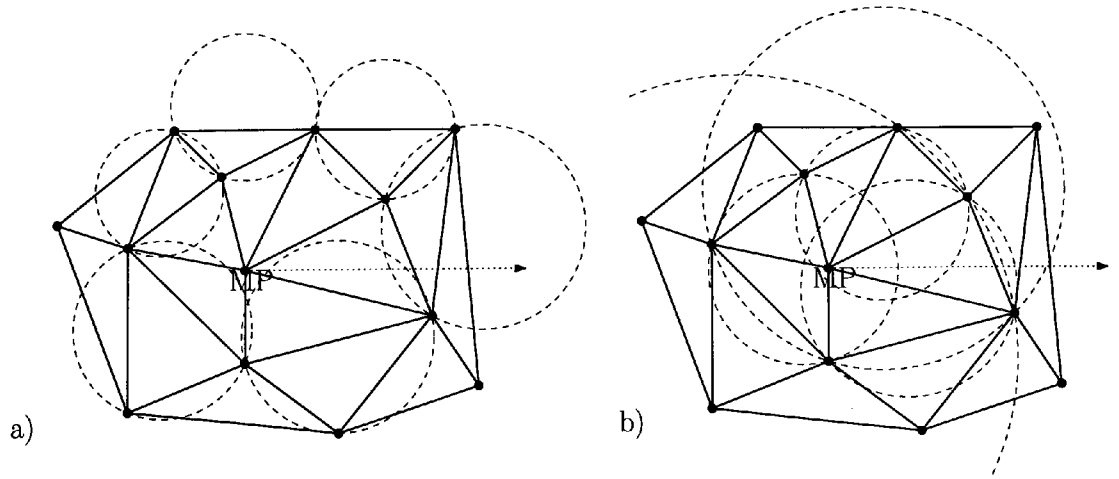


Figure 4.13: Two kinds of circles processed in the moving point routine. a) The real circles. b) The imaginary circles.

Figure 4.14a shows a situation, when the moving point *MP* is about to enter a real circle circumscribed about the triangle *abc*. The location of *MP* is changed to the intersection point of that circle and the trajectory *mn* and *MP* becomes cocircular with points *a*, *b* and *c*. Then the diagonal *ac* of the quadrilateral is swapped to form a new edge *MPb*, connecting *MP* and *b* (Figure 4.14b), so *b* becomes a neighbour of *MP*.

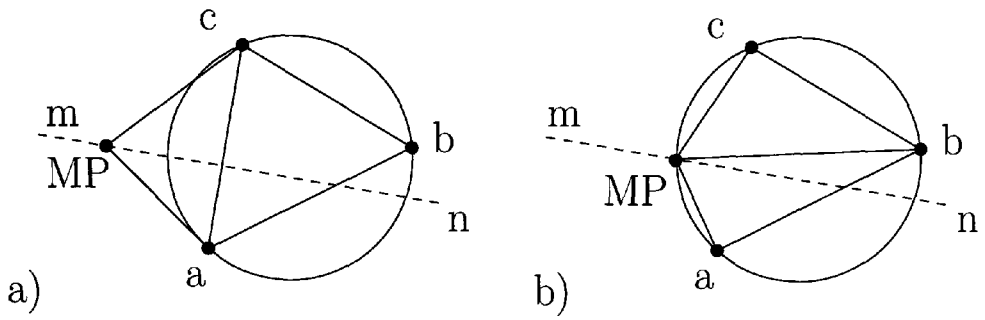


Figure 4.14: The moving point MP enters a real circle. a) Before entering the circle. b) After entering the circle.

Figure 4.15 shows a sequence of the moving point *MP* leaving the imaginary circle *abc*. In the initial step *MP* is located inside the imaginary circle *abc* formed by three consecutive neighbouring vertices of *MP* (Figure 4.15a). In the next step *MP* is moved into the intersection point of the circle *abc* and the trajectory *mn*. The edge connecting *MP* with *a* is swapped, joining vertices *b* and *c*, so *MP* loses the point *a* as its neighbour (Figure 4.15b).

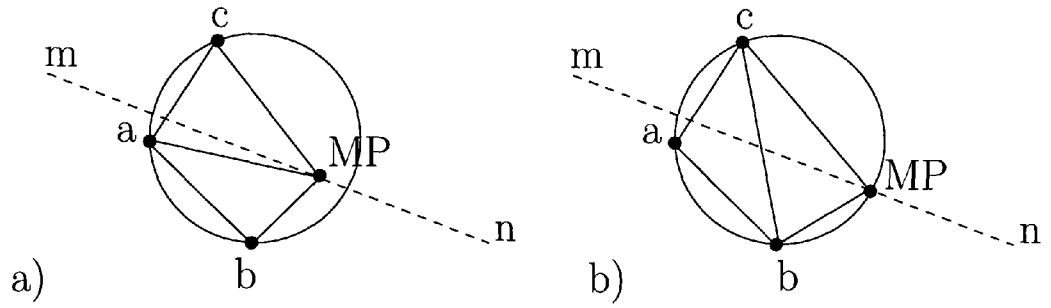


Figure 4.15: The moving point MP leaves an imaginary circle. a) Before leaving the circle. b) After leaving the circle.

This method was extended by Mostafavi and Gold (2004) who used the Voronoi diagram to simulate fluid flow. They proposed a Free-Lagrange solution (Fritts et al. (1985)) for a simultaneous movement of multiple points, so points can freely move in the mesh, as their locations are not fixed. This method is time-step free, so it avoids overshoots and undetected collisions and is based on processing topological events. A velocity vector is attached to each of the points, and they move and interact with each other (which may cause a change of their velocity vectors as well). Instead of using time steps to determine moments when the whole structure is needed to be updated, the VD is updated locally after processing each topological event. In order to move all points simultaneously their first topological events are computed. Then the points are stored in a global priority queue according to the time of their first topological event. Then points are retrieved from the queue, their first topological events are triggered, and the neighbours of moved points and the queue are updated appropriately.

One of the important issues in kinetic data structures, often avoided by researchers, is a point collision. Most assume than moving points do not collide or do not even mention that possibility, in the real situation of points interacting with each other collisions happen very often as their trajectories intersect. There are two issues - how to detect a collision and how to deal with it.

Crowley (1985) discussed mesh optimization and dealt with edges of a triangulation being shorter than a specified length. He proposed moving them away or merging them together. The merge operation replaces two nodes with one in the middle of them, which removes two triangles adjacent to the edge joining those two nodes, as depicted in Figure 4.16. Augenbaum and Peskin (1985) proposed a similar approach - replacing two nodes with an average one if they are close enough to each other. They also discussed collision detection. After determining a new position for a point all its neighbours are tested. If one of them is closer than a specified distance then the points are merged.

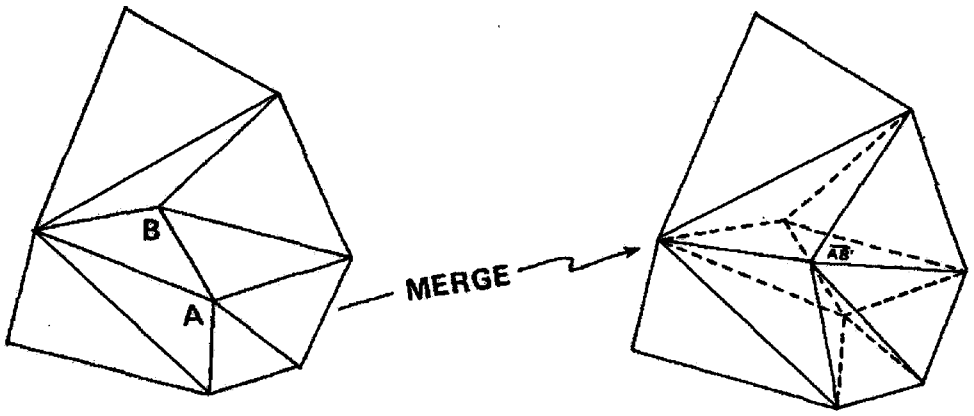


Figure 4.16: Merging two nodes in an arbitrary triangulation. (modified from Crowley (1985))

Mostafavi (2002) proposed a different solution. He assumed that all nodes in the mesh are assigned an equal size as their attribute, so they can be processed like disks. He also assumed that two disks cannot overlap. He discussed collision detection when dealing with a point movement based on topological events. After detecting the nearest topological event defined by one of the circumcircles, the location where the movement trajectory is cut by the circle is tested. A disk is defined there and all neighbouring points are tested for disk overlap. If the defined disk overlaps any of the neighbours it means that there is a collision at the location of that topological event. In such cases he proposed removal of the moving point and its reinsertion at a specified distance on the other side of the collision point.

4.6 The Constrained Delaunay Triangulation Algorithms

The construction and management of the CDT requires algorithms for insertion and deletion of constrained edges and ordinary points. There are several methods for inserting constraints into the DT. The traditional one and probably the most studied method is based on the idea of inserting constraints one by one by deleting triangles crossed by them (Shewchuk (1998)). It is an incremental method, so the set of constraints does not have to be known before the construction and the insertion of additional constraints is possible at any time without rebuilding the whole graph.

Anglada (1997) presents a detailed description of such an algorithm. Both endpoints a and b of the constraint S are inserted first. Then triangles crossing S are detected (Figure 4.17a) by walking from *v1* towards *v2* using adjacency relationships and testing intersections with S. Edges crossing S are removed starting from a position and their endpoints are stored to be used later for retriangulating the hole (in two groups depending on which side of S they are located). After reaching b the edge S is inserted (and marked as constrained) creating polygons on both sides of S (Figure 4.17b). These polygons are retriangulated using two groups of vertices of the intersected edges by testing if the circumcircle of each potential triangle is free of the rest of points of the polygon. The resulting triangulation (Figure 4.17c) is a constrained DT.

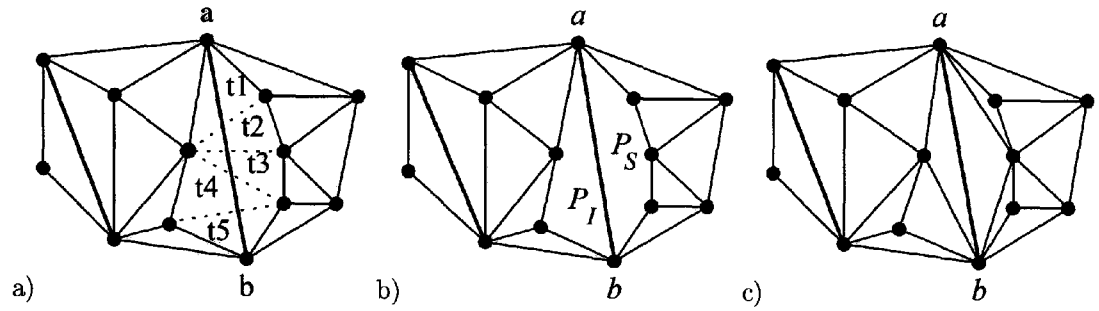


Figure 4.17: Inserting a constrained edge into a CDT. a) Triangles crossing the constraint. b) The CDT after removing marked edges and inserting the constraint. c) The result. (from Anglada (1997))

Kallmann et al. (2003) used the same method to insert constraints but extended it to handle intersecting constraints. In his method, each existing constraint intersecting the newly inserted one is split at the intersection point by inserting an additional point. He also presented a location mechanism, which is an important part of incremental methods. He modified the DT visibility walk for the CDT, by marking visited edges while walking.

Sloan (1993) proposed a slightly different approach for adding constraints, where triangles intersecting inserted segments are not deleted, but transformed by swapping edges until the constraint appears in the triangulation. In his method the triangulation is complete at each step of the algorithm. A similar method was described by Bernal (1995). More recently Shewchuk (2003) proposed a method of constructing and updating the CDT using bistellar flips.

Another group are batch algorithms where all constraints are used to build the CDT and the insertion of a new segment requires restarting the whole process. One well known one is a divide-and-conquer method by Chew (1987). It is a fast and complex approach running in the optimal $O(n\log n)$ time. Lee and Lin (1986) constructed the CDT in $O(n^2\log n)$ time. Seidel (1988) used a plane-sweep method to construct the CDT in $O(n\log n)$ time.

The deletion of constraints has been rarely tackled by researchers, who rather focus on the insertion process. Kallmann et al. (2003) proposed a dynamic CDT model where constraints could be moved by deletion and insertion at another location. In their work the constraints can be complex objects, consisting of several connected segments, and intersections are also allowed. The deletion process starts with locating one of the final endpoints of the constraint. Then all segments forming the constraint are located (Figure 4.18a) and set to unconstrained by changing their flag values (Figure 4.18b). In the next step all endpoints of constraints and points of intersection with other constraints are deleted by removing all edges adjacent to deleted vertices and retriangulating resulting polygons (Figure 4.18c).

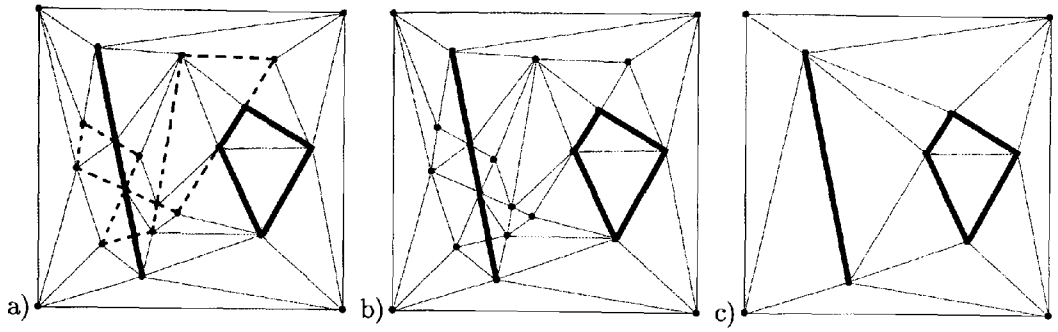


Figure 4.18: A constraint deletion from the CDT. a) Constrained edges marked for removal (dashed lines, intersecting other constraints). b) The CDT after removing constraints. c) The CDT after removing remaining endpoints and intersection points. (from Kallmann et al. (2003))

The insertion of ordinary points into the CDT can be performed using the DT incremental method with one modification, that constrained edges cannot be flipped during the Delaunay optimization process (Shewchuk (1998)). Point deletion is usually done by removing all edges adjacent to the deleted node and retriangulating the resulting polygon (Kallmann et al. (2003)).

4.7 The Line Segment Voronoi Diagram Algorithms

The Line Segment Voronoi Diagram (LSVD) construction has been studied since the beginnings of Computational Geometry. Unfortunately most of the papers (cited below) provide very brief and vague descriptions of algorithms, usually without any implementation details, making the practical use of them almost impossible. This, and the complexity of the problem itself, are two main reasons why there are a very few implementations of the LSVD available. Also I have not been able to find any papers referencing to deletion of line segments, nor points in the LSVD.

Most of the first LSVD construction methods were batch algorithms, producing the result from all data in one operation. One of this was the sweepline method of Fortune (1987) to construct a Voronoi diagram of line segments in $O(n \log n)$ time. Those algorithms will not be discussed further, as they do not allow interactive modification of the mesh.

Imai (1996) and Sugihara et al. (2000) proposed a topology-oriented incremental approach of inserting line segments into the VD. Their works shows that it is possible to design a geometric algorithm in finite-precision environment, by slightly changing the concept of validity of the procedure. Using floating-point arithmetic for implementing geometric algorithms leads to numerical errors and their method values the VD topology correctness and consistency more than the exact results of calculations. No matter how large the numerical errors are, the algorithm always produces a topologically consistent diagram. They assume that segments do not intersect, but they can form not touching polygons (so no point belongs to more than two segments). For a newly inserted Voronoi region *Vr* of a segment *LS* (gray area in Figure 4.19), they assume that vertices and edges of *Vr* form a closed path surrounding vertices of the VD that have to be removed, that vertices and edges to be removed to make a new cell form a tree (thin dotted lines in Figure 4.19) and there is a unique path between endpoints *LP1* and *LP2* of the segment *LS*. Using those assumptions both endpoints of the segment are inserted first. Then searching for a path between *LP1* and *LP2* is performed and a tree sequence of vertices to be removed from the VD is constructed. In the next step a new Voronoi region of *LS* is constructed - the new Voronoi vertices surrounding the tree

are inserted and the tree vertices are removed. The result of their method may have numerical imperfections, but still is usable for many applications.

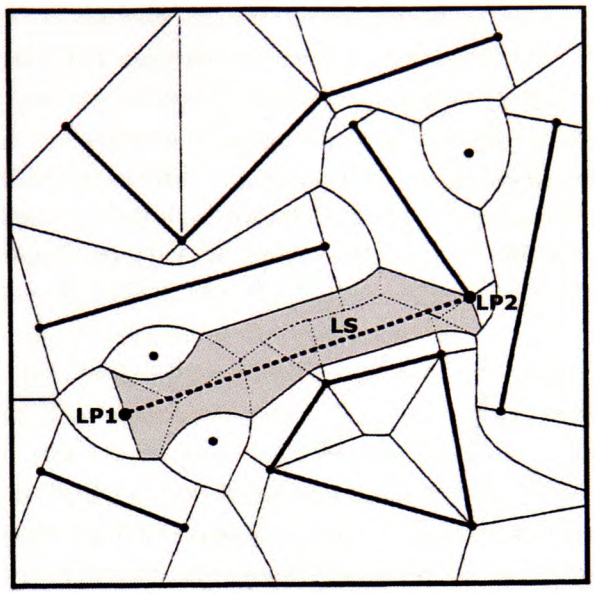


Figure 4.19: Adding a segment LS into the LSVD (modified from Imai (1996)).

Held (2001) extended and implemented the topology-oriented algorithm, developing a package called VRONI for computing the LSVD of points and segments. It is based on floating-point arithmetic and is one of the few known operationally stable implementations of the LSVD. The main focus of his work was to produce a completely reliable and fast in practice algorithm and it took about 10 years to accomplish that. His method is based on "four corner stones": the topology-oriented approach of Imai (1996) and Sugihara et al. (2000) for any possible combinations of segments and points; a very careful implementation of the numerical computations required; an automatic "relaxation of epsilon thresholds"; and a multi-level recovery process combined with a "desperate mode" for special handling of degenerate cases. In his method, having a set of points and line segments, a Voronoi diagram of all points together with the endpoints of line segments is created. In the next step the input line segments are inserted incrementally between existing sites of the VD using the method of Sugihara et al. (2000). The validity of the diagram was tested after each step of the algorithm and in case of the diagram failing topological or numerical checks the last operation was repeated using a different precision value until the diagram was correct ("relaxation of epsilon thresholds"). If none of the precision values lead to a valid diagram a "desperate mode" was applied, where all numerical tests were neglected in order to produce a topologically correct "best possible" diagram. Held (2001) paper ends saying that extensive experimental tests on synthetic and real-world test data have shown that VRONI is reliable.

The latest approach to the LSVD construction was proposed by Karavelas (2004), and this method is used in the CGAL library. It is an incremental method using exact arithmetic, handling intersecting segments and using a Voronoi hierarchy (consisting of a set of Voronoi diagrams) for location purposes. He defines two kinds of site intersection: sites intersect weakly if their common point does not lie in the interior of any of those sites (black intersection marks in Figure 4.20a)

and sites intersect strongly if their shared point lies in the interior of at least one of the sites (white intersection marks Figure 4.20b). He presents two insertion methods for both kinds of intersecting sites. For weakly intersecting objects the line segment LS insertion process consists of four steps: finding the first "conflict" of LS with the VD, finding the entire "conflict" area, constructing the new VD with LS included and updating the location data structure. For strongly intersecting sites, when inserting a point onto a segment, the existing segment is split into two pieces joined by the new point and the VD is updated by splitting and adding appropriate Voronoi edges. When inserting a segment intersecting another segment the two endpoints are inserted first, the existing segment is split at the intersection point and two halves of the new segment are added. He describes also a simple walking method, in which the neighbouring sites are tested if any of them is closer to the specified location than the current site. If any of the neighbours is closer the process proceeds to this object, otherwise the current site is the nearest object.

He employs explicit and implicit representations for sites. The explicit representation simply uses coordinates of nodes. The implicit one encodes the history of site creation and utilizes the fact that the intersection point is a common point of two input segments, and segments that are not among the input data are results of operations on input segments. There are five representations in total - two for points and three for segments. Points are represented by their coordinates, or in case of intersection points of strongly intersecting segments, by the endpoints of those segments. Three representations of segments are combinations of endpoints, input segments (for intersecting segments) and Boolean values for additional processing (see the paper for the details). Such a representation increases the robustness of the method by avoiding "an exponential blow up on the bit complexity of the coordinates of the points of intersection" in case of the input data having many intersections. Along with this representation he uses a "geometric filtering" technique to decrease the number of times the exact arithmetic computations are performed, by applying logical operations on sites or trying to use floating point arithmetic if possible, instead of costly exact operations.

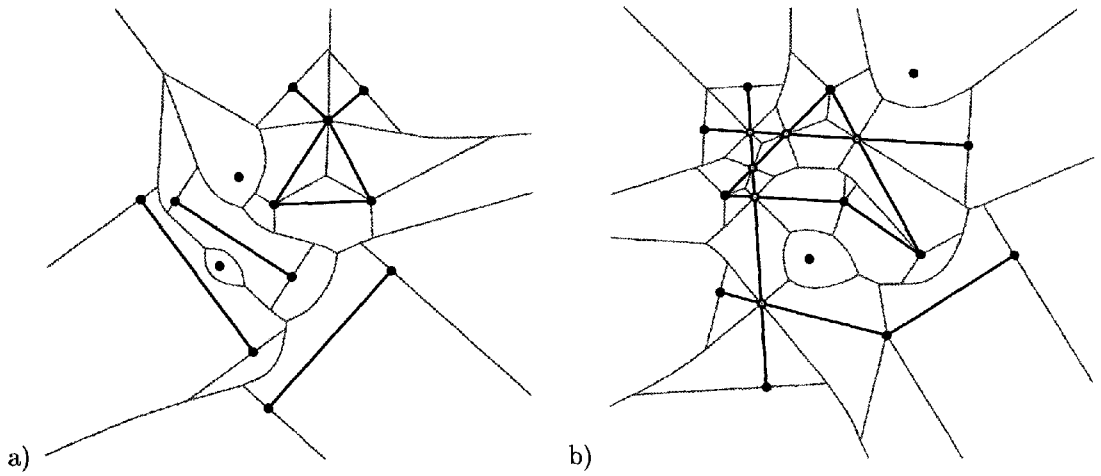


Figure 4.20: Two types of site intersections. a) Weakly interesting sites (black intersection marks). b) Strongly intersecting sites (white intersection marks, from Karavelas (2004)).

A completely different approach for the LSVD construction was introduced by Gold (1990) who proposed inserting a line segment by splitting a new point from an existing node, converting the edge between these two nodes into a new segment and expanding it towards the other endpoint

location. The deleting of a segment would be an inverse operation, shrinking the segment towards its other endpoint and merging both endpoints at the end. This method for calculating the LSVD of disjoint sites was further explained in Gold et al. (1995) and by Yang and Gold (1995) and Gold et al. (1997) for segments also connected at endpoints. The method was extended later by Yang (1997) for intersecting segments. It is an incremental and kinetic process of inserting segments into the VD/DT and it is the basis for the work in this thesis.

Segments are inserted using the idea of the moving point in the VD, introduced in Section 4.5. First of the endpoints *LP1* is inserted into the VD and the segment itself is constructed by splitting a new point *LP2* from *LP1* (both exactly at the same location) and converting the new "zero-length" edge between them to the new segment object. The new point *LP2* is moved towards the desired second endpoint location, along a straight line trajectory using previously described moving-point algorithm (modified for circumcircles involving line segments), expanding the new segment and maintaining the correct VD at the same time. The method is based on triggering consecutive topological events, which are changes in topology of the VD. Topological events occur when four objects become tangent to a common circumcircle. Events are triggered by relocating *LP2* to the positions on the trajectory where a change in the topology is needed to maintain the correctness of the diagram. These positions are the intersection points of the trajectory with circumcircles of the diagram and they can be determined by processing the neighbouring sites of *LP2*. The moving endpoint *LP2* is relocated to the nearest position of the topological event and the VD is updated by switching an appropriate Delaunay edge. It adds a new neighbour to the endpoint, or removes one of its neighbours, making it a neighbour of the interior of the segment. The process is repeated until the destination location is reached, so the new line segment "covers" completely the trajectory. This produces a new segment connected to the sites approached in the expanding process. The construction process is reversible and each segment can be deleted by retracting it, which means moving one endpoint towards the other. The order of the processing of topological events is crucial to the robustness of the method. Any error in the computing of event positions may lead to selecting an incorrect event as the nearest and results in a wrong VD.

Figure 4.21a shows a line segment during its construction process that was started from *OP* and is carried towards the *DP* location. Figure 4.21b shows the next stage of the process, after processing one topological event and relocating *MP* closer towards *DP*. The length of the *ve1* Voronoi edge was reduced to zero and the edge connecting *MP* with *p1* was swapped, connecting the interior of the new segment with point *p1*, so *p1* is not a neighbour of *MP* anymore

This method has many advantages - it is dynamic, sites can be added or removed at any time, it is relatively simple to implement, and theoretically it can be extended to handle intersecting segments. The main problem faced by the authors was handling numerical errors caused by using floating point arithmetic, which led to incorrect diagrams on various occasions. Examples include the centre of a circumcircle of three objects determined on the wrong side of one of the objects, or a projection of a circle tangent to a segment at its endpoint falling slightly outside that segment. This research improves on that earlier approach as it employs an iterative circumcircle calculation method of Anton and Gold (1997) (described in Section 5.3.3). This method guarantees that the resulting circumcentre falls on the correct sides of the line segments, and that the vertices and the tangent points of the circumcircle are in correct anticlockwise order, producing better results than the method used in the earlier work

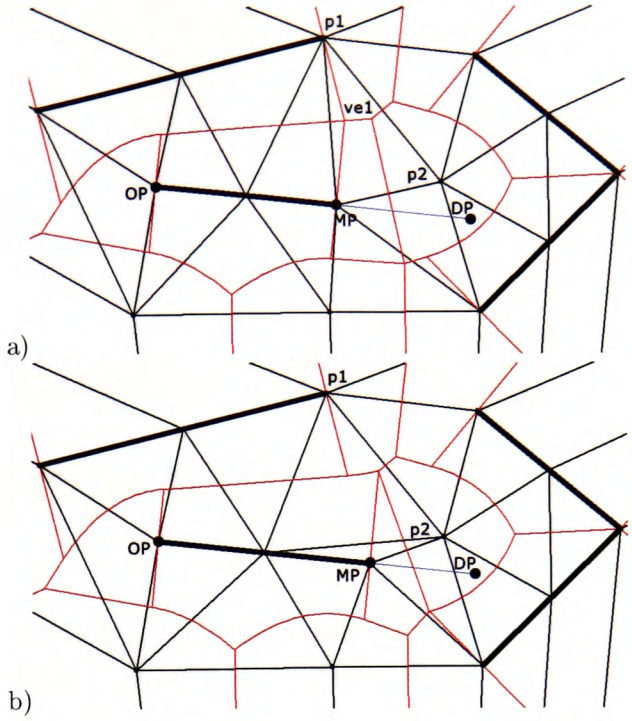


Figure 4.21: The line segment expansion. a) The initial stage. b) The expanded segment after processing one topological event.

Chapter 5

Kinetic Voronoi/Delaunay Construction and Update

This chapter presents the algorithms and data structures (introduced in Dakowicz and Gold (2006); Gold and Dakowicz (2006, 2007); Gold et al. (2008)) used to create and manage the Delaunay triangulation, the Voronoi diagram, the constrained DT and the line segment VD.

The VD algorithms developed and described in this thesis are based on the ideas of the kinetic LSVD construction introduced by Gold (1990), Gold et al. (1995) and extended by Yang (1997), where a new line segment is a trace left by the moving point. Points are added using the kinetic "split and move" method and removed using "move and merge". This is because the well known incremental insertion (Guibas and Stolfi (1985)) and "the ear based" deletion algorithms (Devillers (1998); Mostafavi et al. (2003)) require "straight edge triangles" and thus can not be used for the LSVD construction (where edges connecting points and segments are curved). Due to the complexity of the problem there are very few existing implementations of the LSVD, those known include work by Held (2001) (using floating point arithmetic) and Karavelas (2004) (using exact arithmetic). However, they do not tackle the segment deletion problem, so modification of the diagram is not possible.

The novelty of this work is a refinement of the iterative method of calculating circumcircles of three objects of Anton and Gold (1997) and applying it for calculating Voronoi vertices in the LSVD. Another original idea is creation and manipulation of the Constrained DT using the same kinetic method as for moving points and line segments. All types of meshes are created and managed using the same set of operations. Also there are no restrictions put on the input data - line segments or constrained edges can intersect at any locations and in any possible way.

The VD/DT/CDT/LSVD can be build and modified by the user using a selection of high-level construction operators, such as object insertion or deletion. For these to function a variety of basic tools and low-level operations must be defined.

Basic tools are described in the first part of this chapter (Section 5.3) and include arithmetic and geometric operations, such as distance and circumcircle calculations, orientation and collision tests. Also the mesh traversal and object location problems are included in this group of operators. Most of the operators are rather simple, with the exception of the circumcircle calculation, which is a complex process, so only its idea is described, while all the details are placed in Appendix A.

The low-level operators are the topological construction operators, modifying the mesh. These

operators include Split, Merge and DisconnectLineSegment functions, as well as edge to line segment and line segment to edge conversion operations. All these are explained in the second part of the chapter (Section 5.4). These operators are dependent on the underlying quad-edge data structure operators MakeEdge and Splice, which were described in Section 4.1.

The Move Point operation responsible for relocating nodes, and creating and removing constrained edges and line segments, is hard to classify as low or high-level. It utilizes basic tools such as the distance, collision and circumcircle calculations, as well as low-level operators such as splitting and merging nodes and edge conversion methods. It's complexity is the reason why it will be treated separately. Its idea is provided in this chapter (Section 5.5) and all detail of this complex process are moved to Appendix B.

Another group consists of high-level operators (described in Section 5.6), such as insertion and deletion of objects, which are basically calls of the Move Point operation. These are operations used by the end user to create and manipulate the mesh. These give us our unique dynamic and kinetic approach, where we can insert and delete points, constrained edges and line segments into our map as desired, allowing us to roll the map forward and backward in time.

5.1 Initial Examples

Before explaining all the tools here are some examples of the actual operations. These will help the reader to understand the idea of the kinetic processes and the way they are used to construct and modify Voronoi diagrams.

Every process starts with defining the big triangle in which the insertion and deletion of points is allowed. The range of the data (the square inside the big triangle in Figure 5.1a) has to be known in prior to use it to determine the coordinates of the vertices of the big triangle (see Guibas and Stolfi (1985); Bossen (1996)).

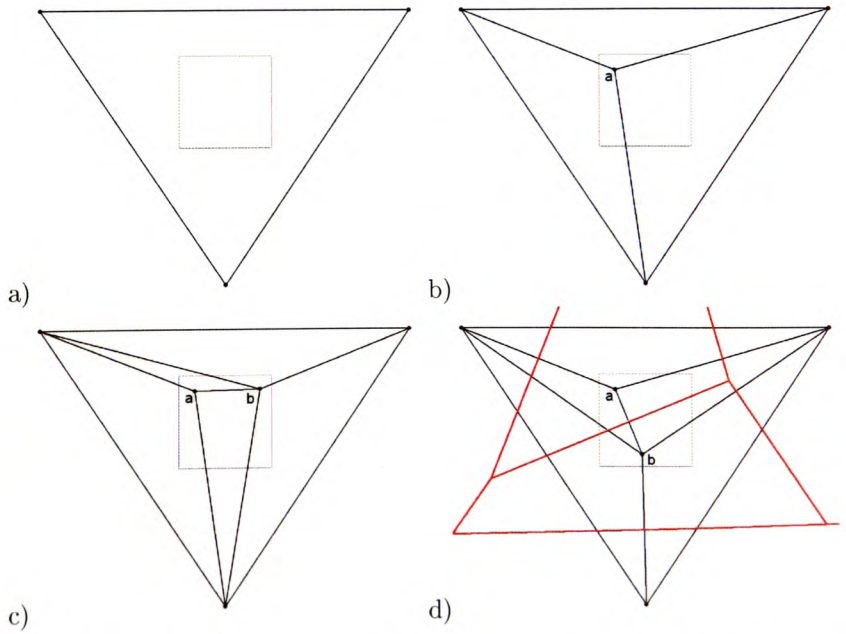


Figure 5.1: Point operations. a) The initial big triangle. b) After inserting one point a. c) After inserting another point b. d) After moving the point b (the Voronoi Diagram is drawn as well).

Then the first point a is inserted, connected to vertices of the big triangle (Figure 5.1b). In the next operation another point b is inserted (Figure 5.1c), by splitting it from a and moving towards the intended destination. In the fourth step the point b is relocated to a new position (Figure 5.1d).

The other set of figures presents operations on line segments. Firstly, a new line segment bc is constructed from the previously inserted point b. It is done by splitting a new point c from b, converting the edge between them into a new segment and expanding the segment towards the intended destination (as will be described in further sections of this chapter), which produces a result shown in Figure 5.2a. In the next operation another segment cd is inserted splitting a new point d from c and moving it towards the intended destination (result in Figure 5.2b). Similarly a third segment de is inserted. However, during its expansion another segment bc is detected on the movement trajectory, so bc at the intersection point u location is split into two segments bu and uc and then two new segments du and ue are created (result in Figure 5.2c). In the fourth step the segment cd is removed by disconnecting it from the endpoint c and retracting it towards another endpoint d (result in Figure 5.2d).

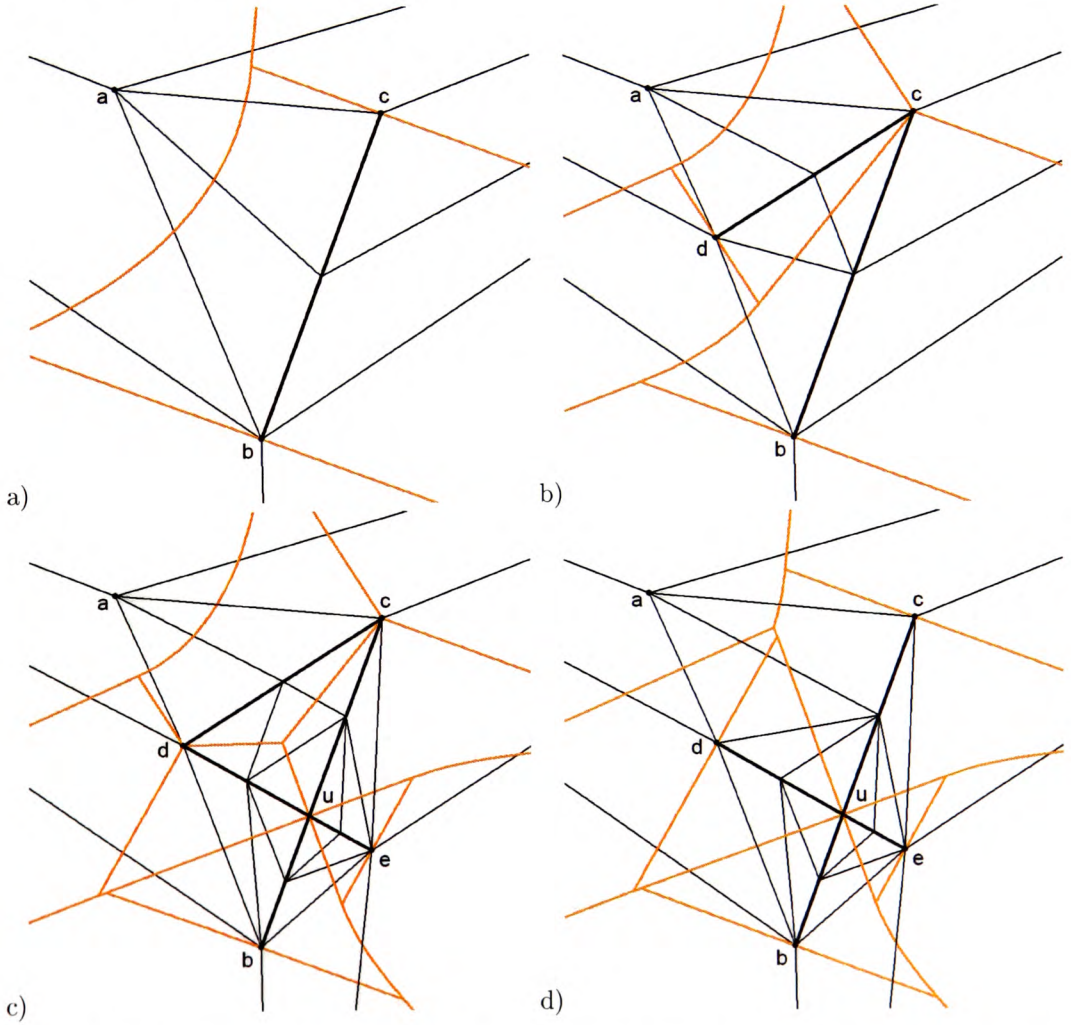


Figure 5.2: Operations on line segments. a) Insert a line segment bc. b) Insert another line segment cd. c) Insert a segment de intersecting another segment bc. d) Remove a line segment cd.

5.2 Object Representations

The ordinary point-based triangular meshes are built of two types of objects - points and Delaunay straight edges. In the CDT there is one additional type of edges - the constrained edge representing input segments. In the LSVD there is one more type of objects - the line segment, and the Delaunay edges connecting line segments and points can be curved, as well as their Voronoi edges which are parabolic.

Point objects in all types of meshes are defined by their x and y coordinates. All edges are represented using the quad-edge structure (Guibas and Stolfi (1985)) in which each edge consists of four Quads. Each quad-edge references a Delaunay edge with its dual Voronoi edge, together with pointers to Delaunay and Voronoi vertices. Incorporating constrained edges into the triangulation requires a slight modification of the quad-edge structure. An additional Boolean flag C for distinguishing between constrained and ordinary edges is added to the TQuad record structure. When C of the edge e is set to true ($e.C := true$ and $e.Sym.C := true$) then e is considered as constrained, otherwise it is a normal edge.

Line segment objects are represented by two oriented half-lines and two endpoints. Each half-line is an object with two references: Org to its starting endpoint and *OppHL* to the opposite half-line.

```
THalfLine = class
  Org:TPointd;
  OppHL:THalfLine;
end;
```

Creating a new line segment *LS* between points *LP1* and *LP2* (see Figure 5.3) consists of making two half-lines *HL1* and *HL2*, setting their endpoints and linking them together. The point *LP1* is set as the starting endpoint of *HL1* and *LP2* as the starting endpoint of *HL2*. In the next step *HL2* is set as the opposite half-line of *HL1*, and *HL1* as the opposite half-line of *HL2*. The resulting line segment consisting of two half-lines is shown in Figure 5.3, together with *Org* and *OppHL* pointers.

```
class function THalfLine.MakeLine(LP1,LP2:TPointd):THalfLine;
  var HL1,HL2:THalfLine;
begin
  HL1 := THalfLine.Create;
  HL2 := THalfLine.Create;
  HL1.Org := LP1;
  HL1.OppHL := HL2;
  HL2.Org := LP2;
  HL2.OppHL := HL1;
  result := HL1;
end;
```

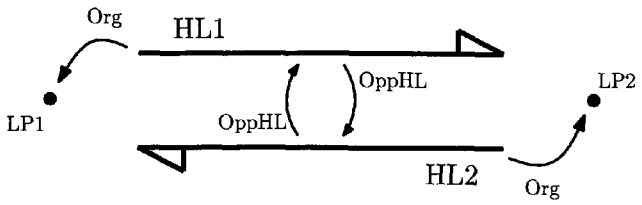


Figure 5.3: A line segment consisting of two half-lines HL1 and HL2 with LP1 and LP2 endpoints.

Incorporating a new line segment *LS* into the mesh between *LP1* and *LP2* locations is a more complex procedure and involves several steps described later in Sections5.6.4 and 5.5.1. Firstly the endpoint *LP1* is inserted. Then *LP2* is added to the mesh at the same location as *LP1*, by splitting *LP2* from *LP1*, which adds three new edges to the mesh. Then the edge between *LP1* and *LP2* is converted to the new line segment (see Section 5.4.4). Finally *LP2* is moved from *LP1* position towards the originally desired location of the second endpoint, expanding the line segment until *LP2* reaches the destination. This adds a new line segment between points *LS1* and *LS2*. All the crucial steps will be described in the following sections. This operation is explained in Section 5.5.1.

5.3 Basic Tools

Basic tools are the basic operations used in low-level operators and in the procedure MovePoint. They include various arithmetic calculations, such as projection and distance, and based on them collision tests. The quad-edge basic operators such as MakeEdge and Splice, described in the previous chapter, can be also classified as basic tools. The complex circumcircle iterative calculation for point and line segment objects is also included in this group. Its idea is introduced in this section, while all the technical details are in Appendix A. Apart from various arithmetic and geometric calculations the mechanisms used to traverse the mesh and locate objects are also included in this group of tools.

5.3.1 Distance Calculation

In the Voronoi diagram of line segments and points the distance calculations are more elaborate than those in the DT/VD/CDT for points only, as there are two kinds of objects in the mesh. Assume we have a point P with x and y coordinates in the plane. We want to compute the distance d from P to one of the objects of the mesh.

If the object is a point PT then the distance d between P and PT is equal to the length of a straight line joining P and PT.

$d = distance(P, PT) = \sqrt{(PT.x - P.x)^2 + (PT.y - P.y)^2}$

If the object is a line segment LS then some additional processing has to be performed. Firstly P is projected onto LS defined by its two endpoints $LP1 = LS.Org$ and $LP2 = LS.Dest$. This returns a value val which is:

- $val < 0$ if the projection of P falls before $LS.Org$, (point $P1$ in Figure 5.4),
- $val \in (0, 1)$ if the projection of P falls inside LS, (point $P2$ in Figure 5.4),
- $val > 1$ if the projection of P falls beyond $LS.Dest$, (point $P3$ in Figure 5.4),

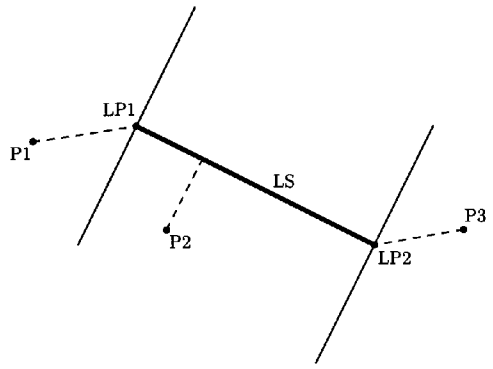


Figure 5.4: A line segment and points at different locations.

Then the distance d is computed according to the result val of the projection:

- if $val \leq 0$ then $d = distance(P, LP1)$
- if $val \in (0, 1)$ then $d = distance(P, LS) = \frac{det(LP1,P,LP2)}{length(LP1,LP2)}$, which is the distance from P to the projection point on LS
- if $val \geq 1$ then $d = distance(P, LP2)$

as depicted in Figure 5.4.

5.3.2 Collisions

Each object in the mesh is associated with a buffer area around it. It is assumed to be equal for all the objects and is stored and accessed by the global *DiskRadius* variable (which can be of thought as a tolerance value). For point objects the buffer is a disk (Figure 5.5a), while for constrained edges and line segments it is an area defined by two disks at the endpoints and buffer zones on both sides of the segment, as in Figure 5.5b.

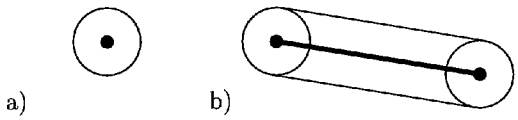


Figure 5.5: Buffer areas of mesh components. a) A point and its disk. b) A constrained edge or a line segment with its buffer area.

When two segments (represented as line segments or constrained edges) intersect their buffer areas intersect as well, as in Figure 5.6a, and a "buffer-zone" like region can be defined, as in Figure 5.6b.

When constructing the mesh collisions of objects often occur. Two objects collide if their buffer areas overlap and this may happen when inserting a new point or relocating a node to a new location. The collision can be detected by finding the nearest object to the tested location and calculating the distance between them. If a collision is detected with two objects then the closest one is selected as the colliding object.

In the case of two points the collision can be detected by testing the distance d between them. If d is smaller than a doubled disk size value *2*DiskRadius* then those two points collide.

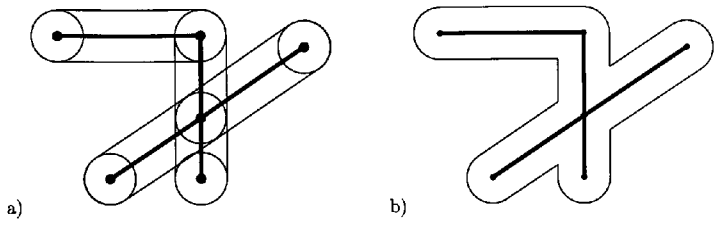


Figure 5.6: Buffer areas of intersecting line segments. a) Disks marked in buffer areas. b) Without making disks.

A collision of a point with a line segment (or a constrained edge) is more complex. If the disk of a point p overlaps a line segment hl it has to be tested if it is a collision with the segment's interior or one of its endpoints. This can be done by projecting p onto the segment and computing the distance d from the projection point to each of the endpoints. If d is smaller than a doubled disk size value ($d<2*DiskRadius$) then p collides with that endpoint, otherwise it collides with the segment interior (point $p2$ in Figure 5.7). The point $p1$ collides with $LP1=hl.Org$ endpoint in Figure 5.7 since the distance from $p1p$ to $LP1$ is smaller than $2*DiskRadius$. The point $p3$ in Figure 5.7 collides with the endpoint $LP2$ and this collision is detected as a collision of two points, since $LP2$ is found as the nearest object to the location of $p3$.

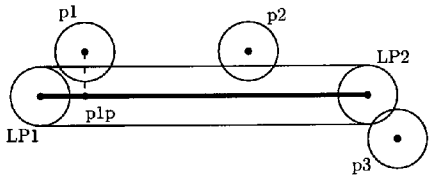


Figure 5.7: Various possible collisions between a line segment and a point.

A collision of two line segments can happen when a new line segments is added using the kinetic method with one of the endpoints $LP1$being moved towards the intended destination, as described later in Section 5.5.1. The collision occurs when $LP1$overlaps a buffer zone of another line segment. Such a collision is handled in the same way as the collision of a point and a line segment. The same applies to constrained edges.

At any static stage of the mesh, when no kinetic operation is performed, there can not be any buffers overlapped with the disks of points. Buffers of connected line segments can partly overlap at the common endpoint, as shown in Figure 5.8a. The only overlapping point disks allowed are those of the intersection points of line segments, created by intersecting two connected line segments with another. This is shown in Figure 5.8b, where two connected line segments $ab$and bcwere intersected by another line segment $de$and the disks of the intersection points $n$and $m$overlap. During the point movement process overlaps of disks are allowed temporarily when the moving point approaches a linear feature on its trajectory, as described later in Section 5.5.1.

5.3.3 The Circumcircle of Points and Line Segments

This section introduces a method of computing the circumcircle of points and line segments. Please refer to Appendix A for a detailed description of this complex process and some additional figures.

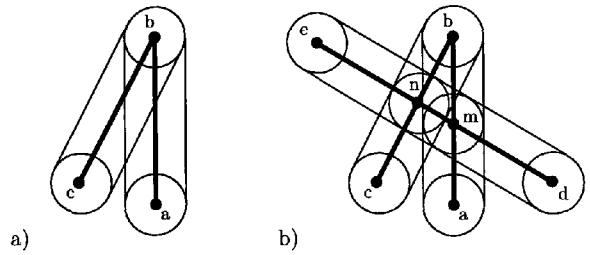


Figure 5.8: Overlapping buffers of connected line segments ab and bc (a) and overlapping disks of intersection points m and n (b).

The circumcircle of a triangle is a unique circle that passes through each of its three vertices (Figure 5.9a). It is defined by the centre point (called a circumcentre) and the radius (called a circumradius). In the Delaunay triangulation of points the circumcircle of each triangle is a Voronoi vertex of the dual Voronoi diagram and can be calculated using simple geometric operations.

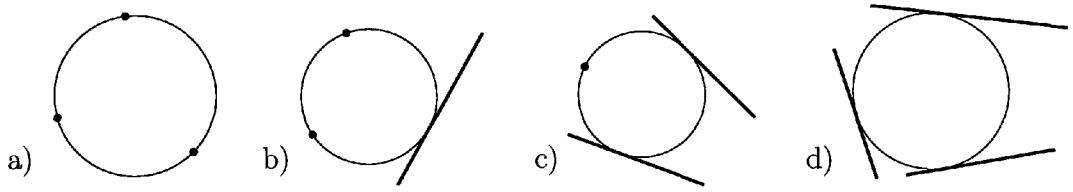


Figure 5.9: Circumcircles of triples of objects.

Computing the circumcircle of a triple consisting of points and line segments (or three segments, Figure 5.9b-d) is a more complex task. Such a circle passes through input points and is tangent to input segments. This can be done using direct calculations by finding the intersection point of the bisectors of pairs of objects (which are lines or parabolas), however this method failed in the earlier attempts of the line segment VD construction (Gold (1990); Yang and Gold (1995)). The problem was that the "arithmetic precision limitations could place the centre on the wrong side of a line segment, thus destroying the node-ordering necessary for topology maintenance" (from Gold and Dakowicz (2006)). The solution selected in this project (see Dakowicz and Gold (2006); Gold and Dakowicz (2006)) is to calculate the Voronoi vertices using the iterative method proposed by Ferrucci et al. (1996) and extended by Anton and Gold (1997), which guarantees finding a correct circle for a ccw oriented triple of objects.

Ferrucci et al. (1996) proposed an algorithm for finding Voronoi vertices for objects of any shape. The only requirement was the ability to find the closest point on each object from a specified point. Starting from a random initial point they find three nearest points (feet) on each of the three objects and compute a circumcentre point for them. This point is used as the next point to find another circle. The process converges to a Voronoi vertex or oscillates with some finite period, depending on the random guess of the initial circle.

Anton and Gold (1997) extended their work and provided a necessary and sufficient condition of convergence. They stated that if the order of points used to calculate the circle is ccw in the initial and further steps, then the circumcentre converges to the Voronoi vertex. In their method, the initial triple of points is randomly chosen until their order is ccw. Then the initial circle passing through those three points is computed. Then the closest point from the circumcentre of that circle is determined for each of the objects and the those three new points (feet) are used to calculate the

next circle, as in the method of Ferrucci et al. (1996). The process is repeated until the distance between the current and the previous circumcentres is smaller than a user-defined tolerance. This produces, for a valid triple of objects, a counterclockwise oriented circle on the correct side of the input line segments centered at the Voronoi vertex location.

However, using directly the foot points on each object to calculate the next circle may lead to a clockwise order of feet for the next iteration (so it would not be be possible to calculate the next circle). This can occur for example when the the projection of the first circumcentre falls outside line segments, as in Figure 5.10. The circle was computed using middle points *C1a*, *C1b* and *C1c* of three segments and the orientation of new feet *CC1ap*, *CC1bp* and *CC1cp* obtained projecting the circumcentre *CC1* onto each of the segment (*CC1bp* falls outside the segment *Pb1-Pb2*) is clockwise. A ccw circle does not exist for these three feet and the iterative process cannot be continued. Such situations do not seem to be handled in the algorithms of Ferrucci et al. (1996) and Anton and Gold (1997) and need to be addressed when working with line segments.

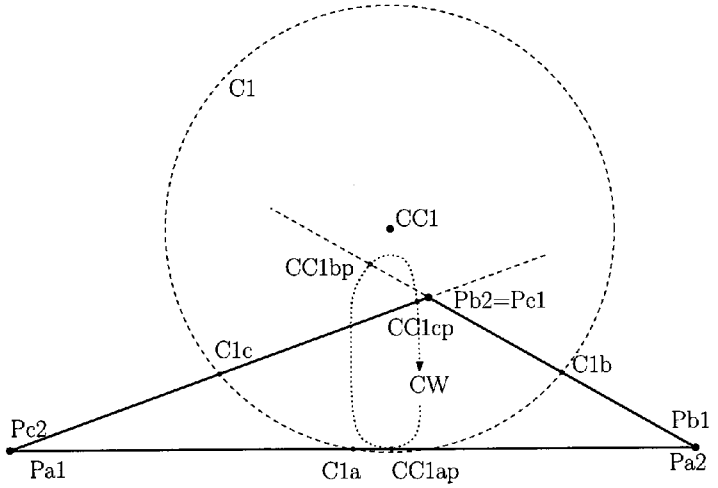


Figure 5.10: Clockwise orientation of the feet.

Also selecting the initial feet randomly often leads to clockwise oriented circles, and in some configurations it may take many attempts to get counterclockwise oriented feet. This is depicted in Figure 5.11, where portions of the segments *a*, *b* and *c* that produce ccw oriented three feet are small comparing to their length. This is the reason why a different method of initial foot selection is desired.

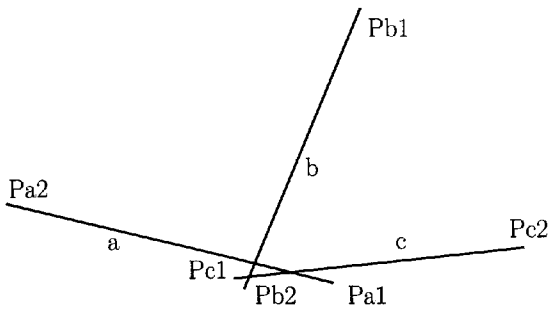


Figure 5.11: A configuration of three segments where the random selection of the initial feet leads frequently to clockwise oriented circles.

The algorithm consists of two main parts. The first is trimming of the input segments to guarantee a ccw order of the initial feet. The second is the iterative process of calculating circles.

The process starts with some initial processing of the input data. Firstly the objects passed to the procedure are reordered to guarantee the same result of the process, by setting their order according to their x, y coordinates (see Appendix A2 for more detail). Due to using floating point arithmetic geometric calculations on a differently ordered set of points can produce slightly different results.

Then, if objects consist of a line segment and its endpoint the circle is calculated using direct geometric operations, as it centre lies on a line perpendicular to the line segment passing through its endpoint. The result is a circle tangent to the line at the endpoint position passing through or touching the third object.

The actual iterative circumcircle process starts with calculating the initial circumcircle, which is computed using the input points or the middle points of line segments. However, using the middle points of the segments directly (even if they are oriented counterclockwise) may lead to a clockwise orientation of the initial points (points *a1*, *b1* and *c1* in Figure 5.12) so a ccw circumcircle would not exist. To avoid such situations a removal of parts of the segments is often necessary.

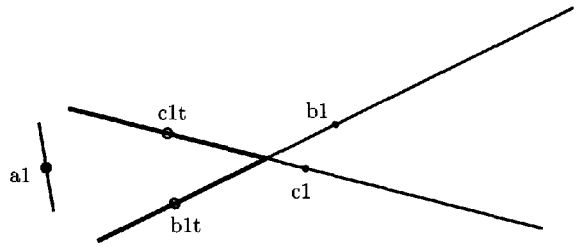


Figure 5.12: Middle points of the original and trimmed line segments.

To avoid clockwise ordered starting points for the initial circle (if any of the input objects is a line segment,) the line segment objects need to be prepared for the iterative processing by trimming them. There are four possible combinations of the three input objects shown before in Figure 5.9: three points (Figure 5.9a), two points and one line segment (Figure 5.9b), one point and two line segments (Figure 5.9c) and three line segments (Figure 5.9d). Each of the combinations is processed separately. A common operation used in each of the three cases with line segments is processing of a pair of two segments. These operations are described below, while additional details and figures are available in Appendix A.

Processing three points does not require any special operations and the circumcircle is computed using the ordinary arithmetic method.

Processing configurations with line segments requires more work. A line divides the plane into two half-planes. For two oriented, infinite and non-parallel lines there can be defined four areas of the plane sharing one common point - the intersection point of the lines. A counterclockwise circle tangent to two lines can only be located in the area being the common "left" area of two lines and for two oriented segments (that can be extended to lines) a counterclockwise circle can be defined if there are at least parts of both segments located along the parts of the infinite lines bordering the common left area of the lines.

When processing two segments it is tested if their configuration makes computing the counterclockwise tangent circles possible and if so to define parts of the segments where such circles

can exist, by trimming unwanted parts of segments. Firstly the intersection point of extended segments is calculated using orientation tests and trimming of the segments is attempted to remove unwanted parts of segments on the right side of the intersection point. Segments are trimmed by changing coordinates of their endpoints to coordinates of the intersection point. As a result none of the segments has any parts on the right sides of each other.

When processing two points and one segment *OBJc*, the segment should be trimmed, so to guarantee the initial circle to be in a ccw order. The trimming can be performed using a temporary segment *OBJt* created from those two input points. Then this pair of segments *OBJa* and *OBJt* is processed, which leads to trimming the input segment *OBJc* with *OBJt* if possible or rejecting the circle.

When processing one point *OBJa* and two segments, the intersection point of two segments is computed (if they are not parallel) and segments are trimmed with the intersection point and the projection of *OBJa* onto them, in a way described in Appendix A.

Processing three segments is performed by processing each pair of them separately. If any of the pairs is not valid then the circle is not possible for this configuration of the segments, otherwise segments are trimmed to their intersection points.

After processing the input data the initial circle *C1* is computed using the input points directly or midpoints of processed line segments (objects a, b, and c denoted here by x and their initial feet by *C1x*). If the order of the input is counterclockwise the resulting circle has a circumcentre *CC1* at *cx, cy* location and its radius is $CR1$, otherwise there is no circle for such a configuration of objects and the process cannot be continued.

The initial circle is then used to obtain a new set of feet for the line segment objects (feet of point objects do not change during the whole process), as shown in Figure 5.13. *CC1* is projected onto each of the line segments. If the projection point *CC1xp* falls inside the segment x then a point half-way between the previous foot *C1x* and *CC1xp* is used as the new foot *C2x* for the next circumcircle (e.g. point *C2a* halfway between *C1a* and *CC1ap* in Figure 5.13). Otherwise, if the projection falls outside the segment then *C2x* is selected between the endpoint of the segment closest to *CC1xp* and the previously used foot *C1x* (e.g. point *C2b* in Figure 5.13). These middle points are used as new feet, because using the projection points directly for the new circle can lead to cw circle results.

In the next step the new circumcircle *C2* is calculated using the new set of feet. It has a *CC2* circumcentre and a *CR2* radius. The distance between the previous circumcentre *CC1* and the new *CC2* is computed. If it is shorter than a specified value *VAL* the process terminates. Otherwise a new set of feet is obtained and the next circumcircle *C3* is computed and compared with the last one until two circumcentres are close enough to terminate the process.

Figure 5.13 shows the process of finding the circumcircle of three connected line segments *LSa* = *(Pa1,Pa2)*, *LSb* = *(Pb1,Pb2)* and *LSc* = *(Pc1,Pc2)*. Firstly three middle points *C1a*, *C1b* and *C1c* of each of the segments are selected as feet and used to calculate the initial circle *C1*. In the next step the circumcentre *CC1* of *C1* is projected onto each segment. Projecting *CC1* onto *LSa* results in a point *CC1ap*, which lies on the segment. Points *C1a* and *CC1ap* are used to compute the next point *C2a* in the middle of them. The second projection point *CC1bp* of *CC1* onto *LSb* falls outside the segment, behind its *Pb2* endpoint, so the new point *C2b* is computed between *Pb2* and *C1b* points. The third projection point *CC1cp* falls inside *LSc* segment so the new point *C2c* is calculated between *CC1cp* and *C1c*. The three new points are used to calculate the second circumcircle *C2* with the circumcentre at the *CC2* location. Using those feet directly

would have led to a clockwise order of points. Then a distance between *CC1* and *CC2* is calculated and compared with a pre-specified tolerance value. Since it is not small enough the circle *C2* is set as the current circle and the new feet for the next circle *C3* are computed projecting *CC2* circumcentre onto each of the segments. This time all projection points fall inside each segment, so middle points between them and previous feet are used. The process ends after n iterations, when two consecutive circumcentres are closer than the specified tolerance value. The resulting circle *Cn* with the circumcentre *CCn* tangent to each of the segments is marked with a solid line in Figure 5.13.

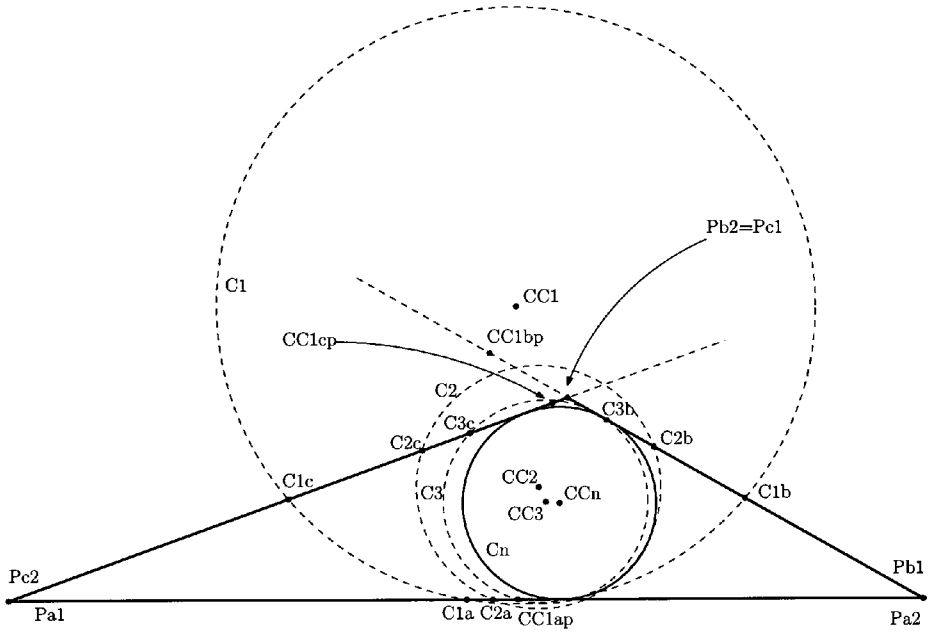


Figure 5.13: Iterations of the circumcircle computing process.

5.3.4 Mesh Traversal

A mesh traversal is used to visit each component of the mesh, preferably visiting each of them once only. The method is used in every operation requiring visiting all components of the mesh and one of the most common usages is mesh drawing. Other examples include rescaling the elevation values of terrain models and runoff modelling using Voronoi cells (see Chapter 6.5.5).

Usually, depending on the type of the mesh a different traversal techniques are utilized. For triangulations Scan (described in Section 4.4) is often preferred, as it is simple, fast and does not require any additional flags or processing. However, this method requires "straight edge triangles" for the orientation test, so can not be used in the LSVD, where edges connecting points and line segments are not straight lines.

The method developed in this research is based on the Depth First Search approach (Sedgewick (1992)) and works for all types of meshes. An additional flag *"visited"* is used in TQuad structure for distinguishing between new edges and those already visited, which reduces the traversal time significantly. An alternative solution is to store visited edges on an additional stack or a list structure, but then testing if an edge was visited would require scanning the whole structure, which is expensive.

The mesh traversal process is explained here (and illustrated by pseudo-code below) on a mesh

drawing example, where every Delaunay and Voronoi edge of the mesh is drawn on the screen. There are two data structures used in the process. The first is a stack *SR* where edges to be processed are stacked. The second is a list *LE* where visited edges are stored in order to quickly reset flags of all edges after traversing the whole mesh. The mesh is traversed edge by edge, and each edge e has two adjacent edges defining a "triangle" (a triple of objects) that is currently processed (*e.Sym.Oprev* and *e.Onext.Sym*) and used to enter neighbouring "triangles".

The process starts from an arbitrary edge *start*, which is pushed onto the stack. Then in a loop edges are popped from the stack *SR* one by one and processed. The popped edge e is marked as visited, added to the list of visited edges *LE* and the edge e and *e.Rot* are drawn. The Voronoi edge *e.Rot* can be a straight line or a parabola, depending on the configuration of the Delaunay objects of its dual Delaunay edge. After processing the edge, its neighbours that have not been visited yet (the value of their *visited* flag is still false) are added to the *SR* stack with a *Sym* orientation. The traversing loop terminates when the stack *SR* is empty. The process ends with resetting the *visited* flag of all edges using the *LE* list.

```
LE := TList.Create;
SR := TStack.Create;
SR.Push(TObject(start));
while true do
begin
  TObject(e) := SR.Pop;
  if e<>nil then
    break
  else
  begin
    e.visited := true;
    LE.Add(e);
    DrawEdge(e);  //Delaunay Edge
    DrawEdge(e.Rot); //Voronoi edge - parabola or edge
    e1 := e.Sym.Oprev;
    e2 := e.Onext.Sym;
    if not e1.visited and not e1.Sym.visited then
      SR.Push(TObject(e1.Sym));
    if not e2.visited and not e2.Sym.visited then
      SR.Push(TObject(e2.Sym));
  end;
end;
for each e from LE do
  e.visited := false;
LE.Free;
SR.Free;
```

Figure 5.14 shows the order of the traversing process of the same mesh for two different starting edges - one being the top edge of the big triangle surrounding the mesh (Figure 5.14a) and another one inside the mesh (Figure 5.14b).

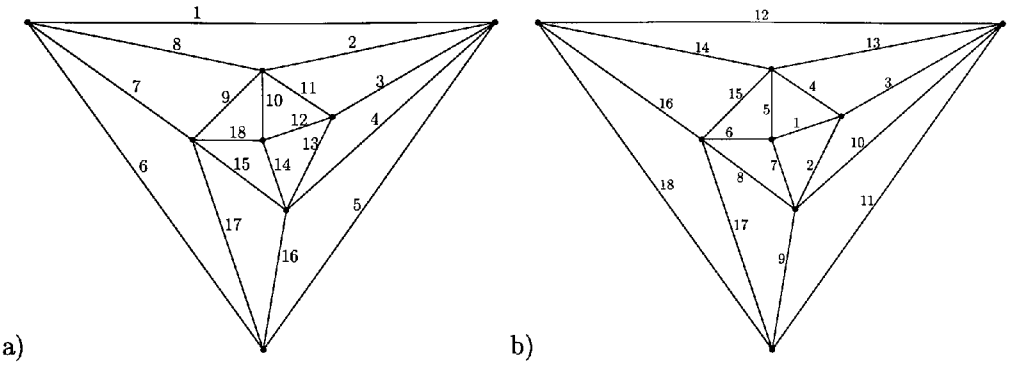


Figure 5.14: Mesh traversal for two different starting edges. a) Starting from the top edge. b) Starting from an edge inside the mesh.

5.3.5 Object Location

The object location operation, often called "Walk", together with the traversal, are the two most used procedures in DT/VD programs. It is used in almost every mesh operation requiring access to a single mesh object. The walk in the DT is used to locate a triangle enclosing a desired location (Devillers et al. (2002)). It is performed by navigating from one mesh object to another using adjacency information. One of the most common usages is to locate a triangle to insert a new point inside it in the incremental insertion algorithm, as described in Section 4.3. The deletion of a point requires its location first as well. The method is also used in most triangulation based interpolation methods.

The visibility walk and other VD/DT methods described in Section 4.3 do not work in the LSVD, as there are edges connecting points and line segments are not straight lines and geometric tests such CCW cannot be used. Another method has to be employed, preferably based on adjacency of objects.

I present a walking method for finding a specified *DP* location based on the Breadth First Search algorithm and working for all three types of meshes - the DT/VD, CDT and LSVD. It in used this research to locate a point from which a new point is split, to test collisions at specified locations or to locate an object for deletion. Its result is an object visible from the *DP* location (so a movement of a point from that object towards *DP* is possible) and as close to *DP* as possible.

The operation consists of two main parts - finding the nearest object to *DP* and then locating a vertex visible from *DP*. The second part is necessary because the nearest located object can be a line segment and in such case one of its endpoints has to be selected as the result. Also, in the CDT the nearest point is not always visible from *DP*. Those cases force some additional processing after locating the nearest object.

The idea of the nearest object location is to start from an arbitrary object *OP* and test if any of its neighbours is closer to the *DP* location than *OP*. If there is one, then the search continues from that neighbour until there are no objects closer to *DP* than the current object.

The process starts from the starting edge *startEdge*, which initially is set to the edge *LastQuad*, which is globally accessible. The origin of *LastQuad* - *LastQuad.Org*, which can be a point or a line segment, is the object from which the location process starts.The distance *mind* from *LastQuad.Org* to *DP* is computed. All neighbours are visited and the distance *d* to *DP* is computed for each of them. If there are any neighbours closer to *DP* than *mind* then the nearest of them is selected. Also if the neighbour is a point and it is at the same distance to *DP* as the currently nearest object

then it is selected as the current nearest, so if a line segment and a point are equally distant to *DP* the point is preferred. The edge *ce* connecting the closest neighbour object and the starting object (with *Sym* orientation) is selected as the new starting edge *startEdge* from which the process continues. The algorithm terminates if none of the neighbours is closer to *DP* than the current object *startEdge.Org*.

However, in the CDT this simple location procedure does not always return the nearest point to the destination location. Figure 5.15 shows a situation where Walk is attempted from the point *OP* towards the location *DP*, and the point *p2* is actually the nearest point to *DP*. After visiting all neighbours of *OP* the point *p1* is found to be closer to *DP* than *OP*. After proceeding to *p1* and visiting all its neighbours none of them is found to be closer to *DP*. A long constrained edge prevents further walk towards *DP* and so the procedure terminates here and returns *p1*, which is not the nearest point (and is not visible from *DP*). To handle such cases some additional processing is performed after finding the "supposed nearest point" *p1* to test if it is really the nearest point. For each object adjacent to *p1* all neighbours are tested, if any of them is not closer to *DP*. For example for *p1* in Figure 5.15 all neighbours of the four neighbouring points (*npla, nplb, nplc, OP*) are tested. If any of them is found to be closer to *DP* than *p1* (*p2* being a neighbour of *npla*, closer to *DP* than *p1* in Figure 5.15) then the whole nearest object location procedure starts again from that point. The process terminates if the supposed nearest point is closer to *DP* than all neighbouring points of its neighbouring points.

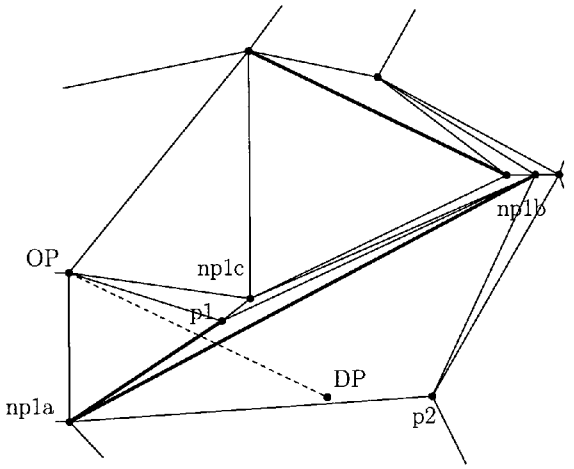


Figure 5.15: Walk in the CDT - the the destination location *DP* is not visible from the reached point *p1* so additional processing is required.

When walking in the CDT, after locating the nearest point some additional processing has to be performed to make sure that *DP* is visible from the located node. Figure 5.16 shows a situation, where *DP* is not visible from the located nearest point *p1* and they are separated by two constrained edges. So after locating the nearest point *p1* to *DP* all edges enclosing *p1* (connecting pairs of neighbours of *p1*) are tested if any of them is located between *p1* and *DP*. If so the same test is repeated for one of endpoints of that edge. The process terminates when there are no edges separating *DP* and the current point *p*, so *DP* is visible from *p*.

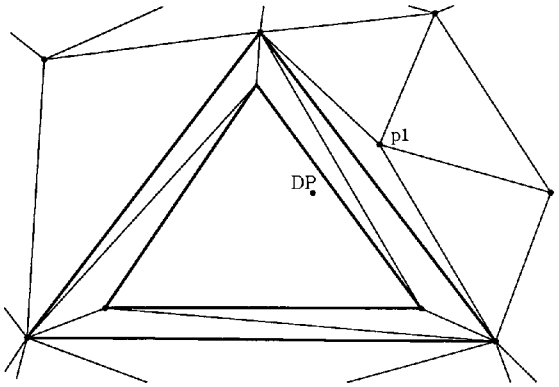


Figure 5.16: The walk towards DP location in the CDT - p1 is the nearest point to DP, but DP is not visible from p1, so additional processing is required.

In the LSVD, if the nearest identified object to *DP* is a line segment (*startEdge.Org* is THalfLine) then it needs to be assured that a correct half-line is selected, so *DP* is on the left side of that line segment (as *DP* might be not visible from the *startEdge.Dest*). Such a situation is presented in Figure 5.17, where Walk towards *DP* was started from the point *P1*. The nearest detected object was a line segment *HL*, or more precisely the half-line *HL1* of that line segment, so the *startEdge* was changed to the edge connecting *P1* and *HL1*. Then all neighbours of the line segment (going around *HL1* and *HL2*) were visited and none of them was found to be closer to *DP* than *HL1*, so *startEdge* with *HL1* as the origin was set as the result of Walk. However, the location *DP* is not inside the Voronoi cell of *HL1* - it is located on the right side of *HL1*, inside the Voronoi cell of *HL2*. So when a line segment is found as the nearest object then additional processing is necessary to assure that *DP* is on the left side of the located half-line, inside its Voronoi cell. A simple orientation test suffices - starting from *startEdge* all Delaunay edges originating at *HL* are visited (*startEdge := startEdge.Onext*) until the orientation of the half-line's endpoints and *DP* is counterclockwise (*CCW(THalfLine(e.Org).Org, THalfLine(e.Org).Dest, DP)>=0*). As the result of this process, a new *startEdge* with its *Org* pointer referring to a half-line whose Voronoi cell contains *DP* inside is set, as in Figure 5.17.

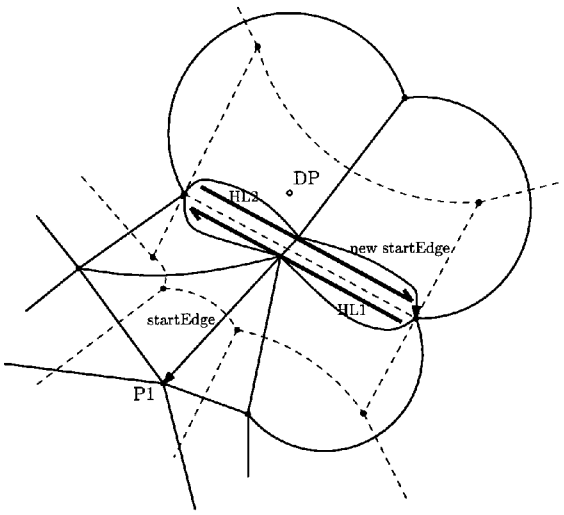


Figure 5.17: The walk process - a special case when the DP location is on the right side of a line segment.

The result of the walk towards the location *DP* operation is an object visible from *DP* and as close to *DP* as possible. This object can be a point or a line segment, but for some operations only points are required as the result, for example when Walk is performed for the Split operation to find a point from which a new point is going to be split. In such cases when a line segment is reported in Walk one of its endpoint is used as the result.

5.4 Low-level Operators

The low-level operators are these used to directly maintain the structure of the mesh. They are called inside the MovePoint procedure to add or delete point and line segment objects. They use basic quad-edge operators such as MakeEdge or Splice to add and remove edges, together with some basic geometric operations, such as the projection calculation. They include the "Split" method for adding a new point to the mesh and the operation "Merge" for removing points by a merging technique. Also the conversion functions used to add and remove line segments ("ConvertEdgeToLineSegment" and "ConvertLineSegmentToEdge") are included as low-level operators, together with a procedure "DisconnectLineSegment" for detaching line segments from nodes.

5.4.1 Split a New Point

In this work, the insertion of a new point in the DT/CDT/LSVD at the location *DP* is not done using the traditional incremental algorithm, but by splitting a new point from an existing node *OP* and moving it towards the *DP*. The incremental insertion works with straight edge triangles only and triples of objects in the LSVD are not always connected with straight edges, so the geometric tests CCW and InCircle cannot be used.

The Split operation adds a new point *MP* to the mesh at the location of one of the existing nodes *OP* of the mesh. The function is used as the first step of the point insertion process, where a new point *MP* is going to be added to the mesh at the location *DP*. This is done by splitting *MP* from an existing node *OP* (found after performing Walk towards *DP*) and moving it towards *DP*. Split is also used in the construction of a new constrained edge or a line segment feature, where the split point *MP* becomes later the moving endpoint of the segment. It adds a new point to the mesh, which becomes the moving endpoint for the new feature.

Let us start the description of the method with a figure presenting the result of the operation, after splitting *MP* from *OP*. Figure 5.18a shows the actual view of the mesh with two points *OP* and *MP* at the same location, which seems to differ from the mesh before splitting *MP* only by the additional Voronoi edge *v01-v02* perpendicular to the trajectory. However, Figure 5.18b shows the actual topological connections in the mesh with points *OP* and *MP* drawn at different locations for the clarity of the image, and with the three new edges *en0*, *en1* and *en2* marked.

The insertion of a new point *MP* into a triangulation adds three new triangulation edges. In the incremental point insertion algorithm these three new edges are inserted inside the triangle containing the new point, connecting *MP* with the three vertices of the triangle. In Split the new point is positioned exactly at the existing vertex *OP* location (as in Figure 5.18a), so selection of the three triangulation vertices to be joined with *MP* is more complex. Obviously *OP* is one of the vertices to be connected with *MP*, by a "zero-length" edge *en0*. The adding of two remaining edges requires more processing, which is based on the fact, that the insertion at the existing point *OP* location splits the Voronoi cell of *OP* into two halves exactly at the position of *OP*. This does

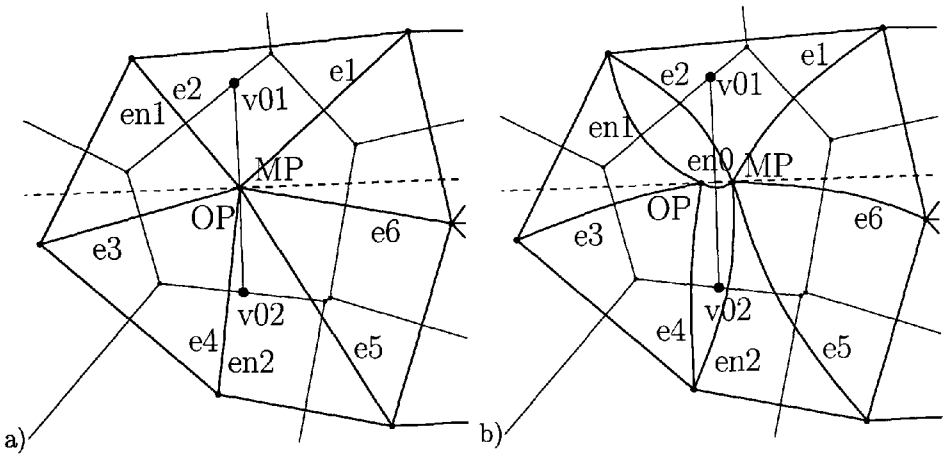


Figure 5.18: The result of the Split operation. a) The actual view of the mesh with nodes OP and MP at the same location. b) The mesh with marked topological connections between OP and MP.

not affect any of the neighbouring Voronoi cells and only introduces two new Voronoi vertices *v01* and *v02* on the boundary of the Voronoi region of *OP*. Each Voronoi vertex is a circumcentre of a triangle, so adding two new Voronoi vertices means adding two new triangles to the triangulation, obviously adjacent to *en0* in this case.

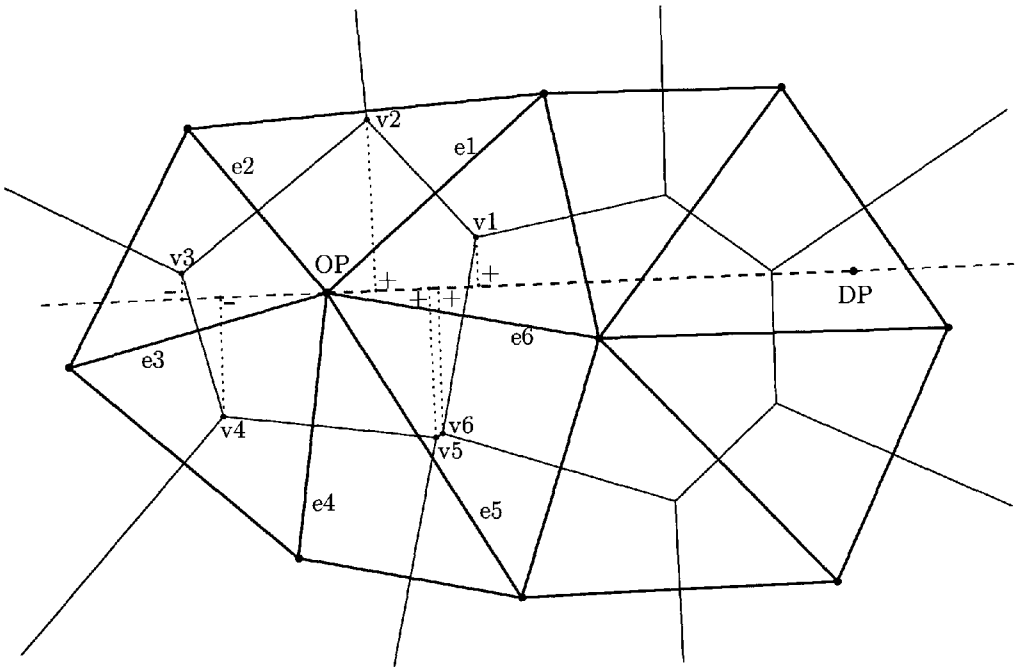


Figure 5.19: Processing Voronoi vertices for splitting a new point.

In the Delaunay triangulation each Delaunay edge has a perpendicular dual Voronoi edge. Because the edge *en0* has both endpoints at the same location it is impossible to determine the orientation of its dual Voronoi edge splitting the Voronoi cell of *OP* into two parts. This is why the trajectory *OP-DP* of the point movement is used to determine the coordinates of the Voronoi vertices *v01* and *v02* of the dual edge of *e0*, which are perpendicular to the trajectory. If we add a Voronoi edge perpendicular to the trajectory at *OP* point location in Figure 5.19 it will cross two

Voronoi edges and *v4-v5* of the cell *OP*. The Voronoi edge *v2-v3* is associated with *e2* edge, and *v4-v5* with *e4* edge. Those two edges will be a part of two new triangles adjacent to *en0* and two additional edges that have to be inserted should join *en0* vertices with other endpoints of *e2* and *e4*. This looks like duplicating those two edges, as *en0's* vertices are at the same location, but is necessary to provide the topology when *MP* moves.

Figure 5.19 shows the process of selection of two edges to be duplicated. *OP* is the origin point, from which the new point *MP* is going to be split. *DP* is the destination point, where *MP* is going to be relocated after splitting from *OP*. The line joining *OP* and *DP* is the trajectory along which *MP* is going to be moved. The point *OP* has six neighbours connected with it by six counterclockwise ordered edges *e1-e6* (the selection of the first edge is arbitrary). Each Delaunay edge has a dual perpendicular Voronoi edge associated defined by two Voronoi vertices, for example edge *e1* has a dual Voronoi edge with *v1* and *v2* Voronoi vertices. Each of those six Voronoi vertices is projected onto the trajectory. The result is a positive value if the projection point is located in front of *OP* (projection points of *v1, v2, v5* and *v6* vertices in Figure 5.19), a negative if the projection point is located behind *OP* (projection points of *v3* and *v4* vertices in Figure 5.19) or equal to zero if projection falls exactly onto *OP*. The edge *e2* is the only edge having a positive value *t1* of its first Voronoi vertex projection and a negative value *t2* of its second Voronoi vertex. On the other hand, the edge *e4* is the only edge having a negative value of its first Voronoi vertex projection and a positive of the second. Projection values for pairs of Voronoi vertices of all other Delaunay edges have the same positive or negative value. As the result the edge *e2* is selected as the Front-Back edge *eFB*, and *e4* as the Back-Front edge *eBF*. Voronoi edges of those two edges will be split into two parts with additional Voronoi vertices after splitting a new point.

```
for each edge et connected to OP do
begin
  t1 := project et.Rot.Org onto (OP, DP);
  t2 := project et.Rot.Dest onto (OP, DP);
  if (t1>0) and (t2<=0) then
    eFB := et
  else
  if (t1<=0) and (t2>0) then
    eBF := et;
end;
```

After selecting edges *eFB* and *eBF* to be duplicated a new point *MP* can be added to the mesh at the exact location of *OP*. This can be done by creating two new triangles having *OP* and *MP* as common vertices, by reconnecting some existing edges and adding three new edges. Firstly, the space has to be made topologically for the new triangles by disconnecting *eBF* and *eFB* edges using the operation Splice(*eFB, eBF*), which separates topologically *eFB* and *eBF*, creating two separate loops around the origin of *eFB* and *eBF* edges. This introduces a non-triangular region to the triangulation which has to be be divided into triangles again. The pointer Org of all edges previously connected to *OP* still refers to *OP* (*eBF.Org* = *eFB.Org*). For the mesh configuration in Figure 5.19 the loop for *eFB* consists of *e2* = *eFB, e5, e6* and *e1* edges (visiting them in the counterclockwise order), and the loop for *eBF* consists of *eBF* = *e4* and *e3* edges. The non-triangular region is defined by edges *e2, e2.Sym.Oprev, e3.Sym, e4, e4.Sym.Oprev* and *e5.Sym*.

In the next step three new edges *en0, en1* and *en2* are created and incorporated into the triangulation using Splice operators.

```
en1 := TQuad.MakeEdge(eBF.Org, eFB.Dest);
en2 := TQuad.MakeEdge(eFB.Org, eBF.Dest);
en0 := TQuad.MakeEdge(eFB.Org, eBF.Org);
TQuad.Splice(en0, eFB);
TQuad.Splice(en0.Sym, eBF);
TQuad.Splice(en1, en0.Sym);
TQuad.Splice(en1.Sym, eFB.Lnext);
TQuad.Splice(en2, en0);
TQuad.Splice(en2.Sym, eBF.Lnext);
```

Then the origin pointer *Org* of all edges connected to *eFB* is changed from *OP* to *MP*. In the last step the Voronoi Diagram is updated. Two Voronoi vertices perpendicular to the trajectory are computed and assigned to two new triangles, which divides the original Voronoi cell of *OP* into two halves with a Voronoi edge separating *OP* from *MP*.

In the LSVD there are cases when for a specified trajectory all Voronoi vertices of *OP* are located in front or behind *OP* and the selection of edges to be duplicated has to be performed in a different way. This often happens for example when a new point *MP* is split from a point connected to the interior of a line segment *LS* to be moved away from there, as in Figure 5.20a. In this case all Voronoi vertices of the Voronoi cell of *OP* are positioned in front of *OP*. If *DP* was located somewhere in the direction of the line segment *LS* all Voronoi vertices would be located behind *OP*.

Dividing the Voronoi region of *OP* with a Voronoi edge perpendicular to the trajectory at the *OP* location creates two cells *c1* and *c2*. The cell *c1* which is closer to *LS* has only two neighbours - *LS* and the point inside *c2*. The cell *c2* is adjacent to *c1*, to all previous neighbours of *OP* and additionally to *LS* on both sides of *c1*. Adding a new node requires adding three new edges, and looking at the divided cell it is easy to configure them. One zero-length edge connects cells *c1* and *c2*, another connects *c2* with *LS* and the third one connects *c2* with *LS* as well but on the other side, as in Figure 5.20b. If projections of all Voronoi points onto the trajectory are positive then the new point *MP* is located in the cell *c2* while *OP* stays in *c1*, if all are negative then *MP* is in *c1*.

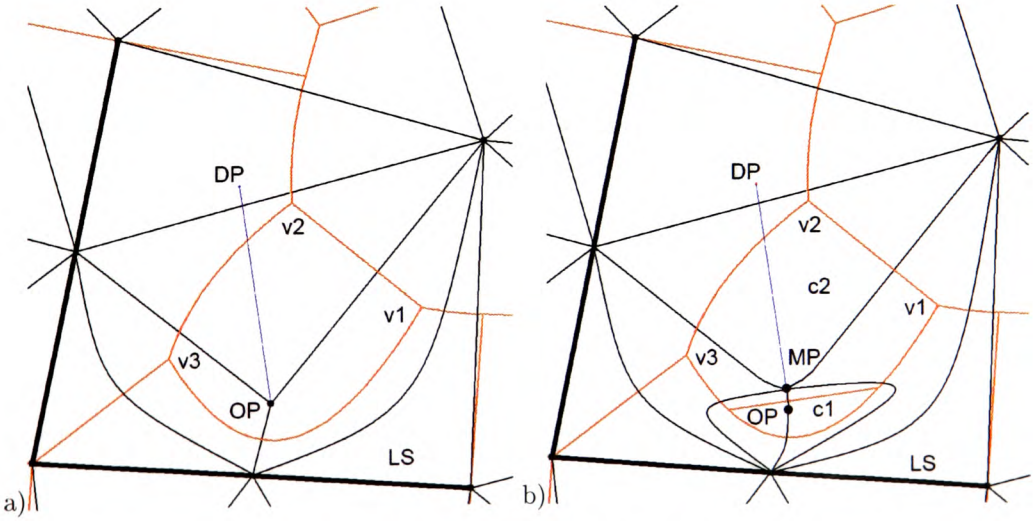


Figure 5.20: Splitting a new point when values of projections of all Voronoi vertices are positive. a) The initial configuration. b) After the Split operation.

In this case two "triangles" are also added to the mesh, however only one edge is "duplicated"

- on both sides. Selection of this edge is straightforward using the orientation test for each pair of consecutive Voronoi vertices and the point *OP* and selecting the edge *e* for which CCW*(e.Rot.Org, e.Rot.Dest, OP)<=0*. Then *e* is disconnected from other edges originated at *OP*, a new point *MP* is created and three new edges are inserted originating at *OP*. One zero-length edge joins those two new points and two others join *OP* with *LS* on both sides of *e*. Then two Voronoi vertices perpendicular to the trajectory are computed and assigned to the two new "triangles", splitting the Voronoi cell into two parts. Finally *MP* is set to *OP*. If the trajectory was oriented towards *LS* then *MP* would not be changed to *OP*.

Splitting a new point from an endpoint of one or more constrained edges requires more processing because Voronoi vertices of constrained edges may overlap. After detecting that *OP* is an endpoint of one or more constrained edges the process starts with locating the triangle intersecting or enclosing the trajectory. This can be done by testing edges surrounding *OP* (connecting pairs of neighbours of *OP*) to find the one that has endpoints *p1* and *p2* on different sides of the trajectory. Then the edge connecting *p1* and *OP* (on the left side of the trajectory) is used as a starting edge for looking for *eFB* edge in a counterclockwise direction using the *Onext* operator. The edge connecting *p2* and *OP* is used as a starting edge for looking for the eBF edge in a clockwise direction using the *Oprev* operator. Edges are tested in the same way as before, using the results of Voronoi vertex projections onto the trajectory. However, there is one addition, that if a constrained edge is approached it is automatically selected as *eFB* or *eBF* respectively.

Split is also used for disconnecting constrained edges from their endpoints, which is performed when deleting constrained edges or splitting them into two halves. In such a case *OP* is an endpoint of *CE* and the trajectory lies along *CE*. The destination point *DP* can be the other endpoint of that constrained edge *CE* when deleting *CE*, or can be located somewhere inside *CE* when splitting *CE* into two parts. Finding a triangle containing the trajectory (like previously) does not work, as the trajectory overlaps the common edge of two triangles. The simplest solution is to pass this constrained edge *CE* as a parameter to the Split function (which avoids using the determinant test for collinear points of the constrained edge and the trajectory endpoint), and use it for searching for *eFB* and *eBF* edges around *OP*. Then the first adjacent edge *CE.Onext* is used as the starting edge for searching for *eFB* in the counterclockwise direction, and *CE.Oprev* is used for locating *eBF* in a clockwise direction.

The resulting mesh contains two points *OP* and *MP* at the same location and two new zero-area triangles created with three new edges. Even though two points are at the same location, the topology of the mesh is correct and the new point can be relocated to the new location. There is a Voronoi edge perpendicular to the zero-length edge "separating" *OP* and *MP*. The zero-length edge between *OP* and *MP* (which can be logically extended along the intended trajectory of the point movement) can be converted to a constrained edge or to a line segment.

5.4.2 Merge Two Points

Merging two points (function "Merge") is an inverse operation to Split. The operation is used to remove a point from the triangulation. It is performed when no more movement is possible for the moving point *MP* and the destination point *DP* already exists in the mesh. It is also the final step of a line segment (or a constrained edge) removal operation, after the segment is converted to an edge and one of the endpoints of this edge is merged with another in order to remove the edge. The operation removes one point from the triangulation and also two triangles adjacent to

the edge connecting those two points. As the result two points are merged into one.

Merge is performed if *MP* and *DP* are the opposite vertices of a quadrilateral, as in Figure 5.21a. Ideally they should be connected with a Delaunay edge, but there is also a situation when they are separated, which will be described later. The result of merging them is a removal of three edges and one of the vertices, as in Figure 5.21b.

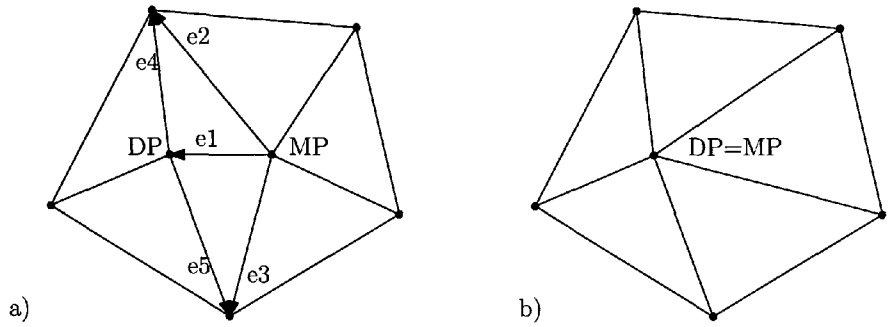


Figure 5.21: Merge operation. a) The initial configuration. b) The result.

The process starts with enumerating edges, from *e1* connecting *MP* and *DP* to *e2*-*e5* for edges defining two triangles adjacent to *e1*. In the next step three edges *e1*, *e2* and *e3* are disconnected from the mesh using Splice operations.

```
//disconnect e1
TQuad.Splice(e1.Sym,e5);
TQuad.Splice(e2,e1);
//disconnect e3
TQuad.Splice(e3.Sym.Oprev,e3.Sym);
TQuad.Splice(e2,e3);
//disconnect e2
TQuad.Splice(e4.Sym,e2.Sym);
```

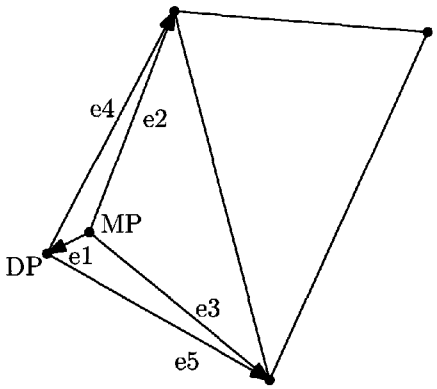


Figure 5.22: Merge operation - a special case with no edges between e2 and e3.

This forms a temporary hole and leaves *e1* and *e3* completely disconnected from the mesh (so for example $e1.Onext = e1$ and $e1.Oprev = e1$). If there were no edges between *e2* and *e3*, as in Figure 5.22, then *e2* is completely disconnected as well, otherwise *e2* is still connected at the *e2.Org* to some edges on its right side. In such a case, depicted in Figure 5.21, *e2* has to be disconnected from its neighbouring edge *e6*=*e2.Oprev* and coordinates of the origins of all those

edges sharing the *e2.Org* endpoint have to be assigned to *DP*. Also, if there is any line segment having *MP* as an endpoint then its coordinates have to be changed to *DP*. Finally, that loop of edges is joined with the loop around *DP* with a single Splice operation, as in the listing below. Then edges *e1*, *e2* and *e3* are deleted and the Voronoi diagram for the neighbourhood of *DP* is updated.

```
if e2.Oprev<>e2 then
//if e2 is not disconnected from the mesh at e2.Org
begin
  e6 := e2.Oprev;
  //disconnect e2 from e6
  TQuad.Splice(e6,e2);
  //for edges around e6.Org point point their Org to DP
  et := e6;
  for each edge et connected to e6.Org do
  begin
    et.Org := DP;
    if (et.Dest is THalfLine)then
      //if MP is an endpoint then change the endpoint to DP
      if MP is endpoint Org of THalfLine(et.Dest) then
        THalfLine(et.Dest).Org := DP
      else
          if MP is endpoint Dest of THalfLine(et.Dest) then
        THalfLine(et.Dest).Dest := DP;
  end;
  //connect e6 and e5
  TQuad.Splice(e6,e5);
end
```

There is a special case, when the the destination point *DP* is the common point of two lines joined at an angle smaller than 180 degrees. If *MP* is connected to two such lines and there is no other object between *MP* and *DP* then there is an edge e connecting those two lines and separating *MP* from *DP*, as in Figure 5.23. This edge can never be swapped because the area of the Voronoi cell of *DP* is zero and *DP* can not become a neighbour of *MP*. The movement of *MP* towards *DP* stops here and the Merge function is called. In Merge the configuration of the mesh is tested around *MP* and after detecting this case the edge e is swapped to join *MP* with *DP*, which temporarily produces an incorrect diagram After this the merging can be performed normally.

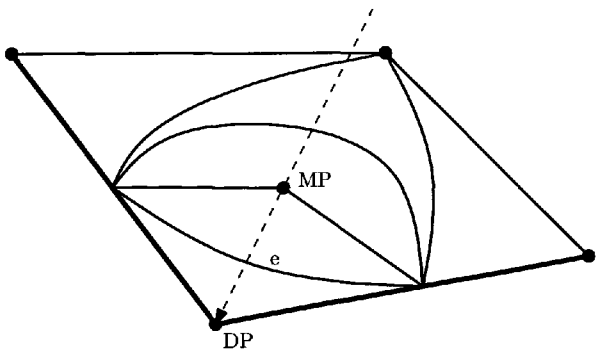


Figure 5.23: Merge operation - merging MP with the junction DP of two line segments.

There are also cases when Merge is called for non-quadrilateral configurations of objects, as in Figure 5.24. The point P in Figure 5.24a is connected to two line segments. A movement is

attempted along the trajectory marked with the dashed line, after splitting a new point *MP* from *P* at the exact location of *P*. A new point *MP* is placed topologically between *P* and the line segment and those objects are its only neighbours. Points *MP* and *P* are in the same location and the length of the edge between them is zero - they have been drawn separately to make the figures more readable. After testing the real and imaginary circles no topological events are detected and since moving points cannot cross line segments, a movement back of *MP* towards *P* is initiated. Then since *MP* and the destination point are identical their merging is called. Then it is detected that *e2=e3* (see Figure 5.24b), so edges *e1, e4* and *e5* are removed and *e2* is joined with the edge formerly adjacent to *e5* on the right side. This situation can happen for different mesh configurations as well.

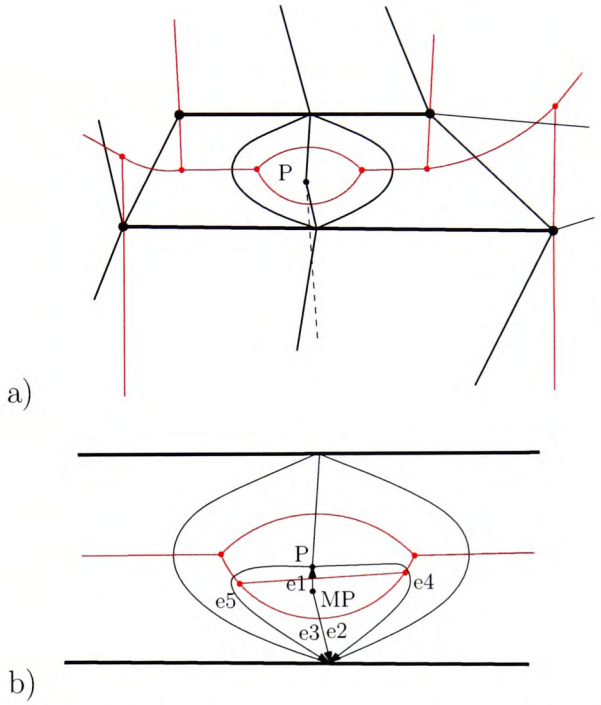


Figure 5.24: Merge operation - the sandwich case. a) The initial situation before splitting a new point from P and moving it "downwards" along the dotted line. b) After MP is split from P and before attempting to move it back towards P (enlarged).

5.4.3 Disconnect a Line Segment from a Vertex

Disconnecting a line segment (function "DisconnectLineSegment") is performed as the first step of the line segment removal process. It detaches the segment from one of its endpoints by adding between them a new edge and point at that location. The function takes the edge having the line segment *HL* as the destination as the argument.

In the first step of DisconnectLineSegment a new point *MP* is created with the coordinates of *P*. Then four edges *e1-e4* adjacent to the line segment are identified, as in Figure 5.25a: *e1 := e; e2 := e.Oprev; e3 := e.Onext* and *e4 := e2.Oprev*.

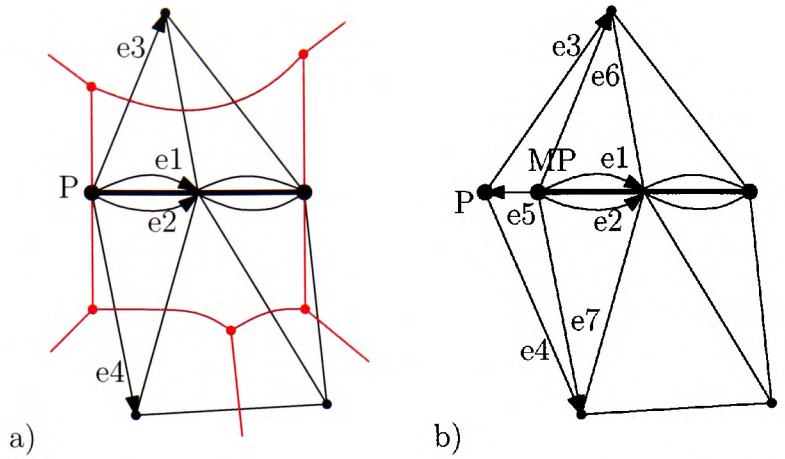


Figure 5.25: Line segment disconnection. a) The initial configuration. b) The topological configuration after splitting MP from P (MP and P are in the same position).

Then the loop of edges around the endpoint P is broken using the Splice operator (calling Splice(*e1, e4*)), disconnecting *e1* and *e2* from *e3* and *e4*. This creates a "topological hole" in the triangulation consisting of six edges. Traversing the hole in the counterclockwise order starting from *e4* produces a sequence: *e4, e4.Sym.Oprev, e2.Sym, e1, e1.Sym.Oprev, e1.Sym* as the result.

In the next step the endpoint of the line segment and origins of edges *e1* and *e2* are changed from P to MP.

Then the hole is split into four triangles making three new edges *e5*-*e7* and incorporating them into the mesh using Splice operations, as in Figure 5.25b. This produces two new thin triangles, having a common zero-length *e5* edge (as MP and P are at the same location).

```
e6 := TQuad.MakeEdge(MP, e3.Dest);
TQuad.Splice(e1, e6);
TQuad.Splice(e3.Sym, e6.sym);
e7 := TQuad.MakeEdge(MP, e4.Dest);
TQuad.Splice(e6, e7);
TQuad.Splice(e4.sym.oprev, e7.Sym);
e5 := TQuad.MakeEdge(MP, e4.Org);
TQuad.Splice(e6, e5);
TQuad.Splice(e4, e5.Sym);
```

Finally the Voronoi diagram is updated for the two new triangles adjacent to *e5* with coordinates copied from the Voronoi vertices of triangles previously adjacent to *e1* and *e2* (*e3.Rot.Org* and *e4.Rot.Dest*), which are still valid after adding those new triangles, because MP and P are in the same position. Also the outer Voronoi vertices of *e6* and *e7* edges (*e6.Rot.Org* and *e7.Rot.Dest*) are updated with *e3.Rot.Org* and *e4.Rot.Dest*.

5.4.4 Convert an Edge into a Line Segment

The procedure of inserting a new line segment into the mesh consists of several steps. Firstly a new point is split from an existing point. This adds a new edge between them that can be used to add a new segment to the mesh, by converting it to a line segment. The conversion inserts two connected half-lines between two existing points and incorporates them topologically into the triangulation. Later this line segment is expanded towards the destined location DP.

The edge to line segment conversion function "ConvertEdgeToLineSegment" is called after splitting a new point *MP* from the existing node *OP*. The Split operation added three new edges to the mesh. The edge e connecting *OP* and *MP* is passed to the procedure as the parameter. Points *OP* and *MP* are at the same location, so the length of e and the areas of two triangles adjacent to e are equal to zero ("thin triangles"). Using this approach greatly simplifies updating the Voronoi diagram after the insertion of the segment.

Converting the edge e to a line segment affects two thin triangles adjacent to e. Figure 5.26a shows a quadrilateral that is involved in the conversion process. Points *OP* and *MP* are at the same location, and the length of edge e is zero, but for the clarity they have been drawn at different locations.

ConvertEdgeToLineSegment starts with making a new line segment. It consists of two half-lines, each originated at one of the endpoints of e (*hl* := THalfLine.MakeLine(*e.Org, e.Dest*)).

This segment has to be incorporated into the triangulation, so in the next step the neighbouring edges *e1-e4* are located and edge e is removed in order to make space for the new segment.

```
e1 := e.Onext; e2 := e.Oprev; e3 := e1.Sym.Onext; e4 := e2.Sym.Oprev;
e.Delete;
```

It creates a quadrilateral hole *OP-RP-MP-LP* in the mesh. Then the new line segment is inserted between *OP* and *MP*. Firstly seven new edges are created with endpoints pointing at the objects of the quadrilateral and the new segment.

```
ne1 := TQuad.MakeEdge(e1.Org, hl);
ne2 := TQuad.MakeEdge(e1.Org, hl.OppHL);
ne3 := TQuad.MakeEdge(hl, e3.Dest);
ne4 := TQuad.MakeEdge(hl.OppHL, e4.Dest);
ne5 := TQuad.MakeEdge(hl, e1.Dest);
ne6 := TQuad.MakeEdge(hl.OppHL, e2.Dest);
ne7 := TQuad.MakeEdge(hl, hl.OppHL);
```

Then the new edges are incorporated into the quadrilateral using twelve Splice operators, which produces a configuration presented in Figure 5.26b.

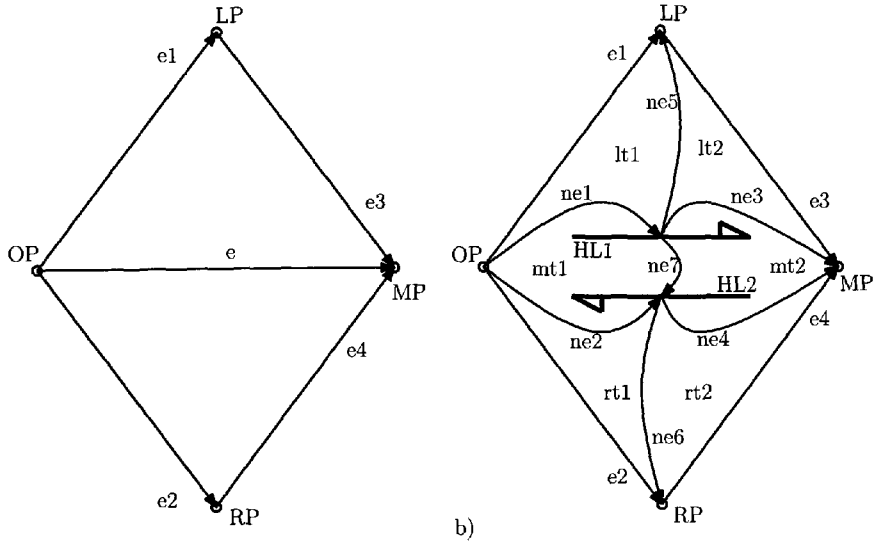


Figure 5.26: Conversion of an edge to a line segment. a) The initial quadrilateral. b) The quadrilateral after incorporating two half-lines.

As the result, three edges *ne1*, *ne3*, *ne5* connect the left half-line *HL1* with three objects of the quadrilateral on the left side of removed edge *e*, three other edges *ne2*, *ne4*, *ne6* connect the right half-line *HL2* with three objects on the right side of *e* and one edge *ne7* connects those two half-lines.

```
TQuad.Splice(e2, ne1);
TQuad.Splice(e1.sym, ne5.Sym);
TQuad.Splice(ne1.Sym, ne5);
TQuad.Splice(ne1.sym, ne3);
TQuad.Splice(e3.Sym, ne3.sym);
TQuad.Splice(ne1.Sym, ne7);
TQuad.Splice(e2, ne2);
TQuad.Splice(ne7.Sym, ne2.Sym);
TQuad.Splice(ne2.Sym, ne6);
TQuad.Splice(e4, ne6.sym);
TQuad.Splice(ne6, ne4);
TQuad.Splice(ne3.sym, ne4.sym);
```

This creates six new triangles, *lt1* and *lt2* on the left side of the segment, *rt1* and *rt2* on the left side of the segment, and *mt1* and *mt2* in the middle of them, all having zero-areas, as the length of *e* is zero initially. In later stages, as the segment expands two left and two right triangles gain some area. The two middle triangles sharing *ne7* edge always stay thin, as the length of *ne7* is always zero.

In the next step the Voronoi diagram has to be updated for the six new triangles. It requires finding six Voronoi vertices and assigning them for each of the new edges and for the edges of the quadrilateral *OP-RP-MP-LP*. Line segments consist of four objects - two half-lines and two endpoints. Each of them has its own Voronoi cell, and Voronoi edges between endpoints and half-lines are perpendicular to half-lines (see Figure 5.27). Since the endpoints *OP* and *MP* of the new segment are at the same location, then pairs of the new Voronoi vertices will be the same positions. Also in order to compute perpendicular Voronoi vertices the orientation of the line segment has to be known. This is why the new line segment's planned endpoint *DP* is passed to the function computing Voronoi vertices as an additional parameter to add some measurable length to the the zero-length line segment. Using *DP* the two locations of perpendicular Voronoi vertices *lvx1,lvy1* and *rvx1,rvy1* are calculated, one for each side of the line segment and their coordinates are assigned for the appropriate new triangles *lvx,lvy* for *lt1* and *lt2* (*v6* and *v5* in Figure 5.27) and *rvx,rvy* for *rt1* and *rt2* (*v2* and *v3* in Figure 5.27). The Voronoi vertices of *mt1* and *mt2* are located at the endpoints of the line segment, so the coordinates of *OP* are used as coordinates of Voronoi nodes for both *mt1* and *mt2* (because *OP* is at the same location as *MP*). As a result six new Voronoi vertices have been added to the triangulation.

Figure 5.27 shows topological configuration of the VD after adding the new line segment, with Voronoi cells for *OP, MP, HL1* and *HL2* and six new Voronoi vertices marked *v1-v6*. For example *v2* perpendicular to the line segment is the centre of *rt1* triangle, having *OP, RP* and *HL2* as vertices. In this figure the line segment has been artificially extended to demonstrate the method, in real both endpoints *OP* and *MP* are at the same location, as are Voronoi vertices *v2* and *v3*, and *v5* and *v6*.

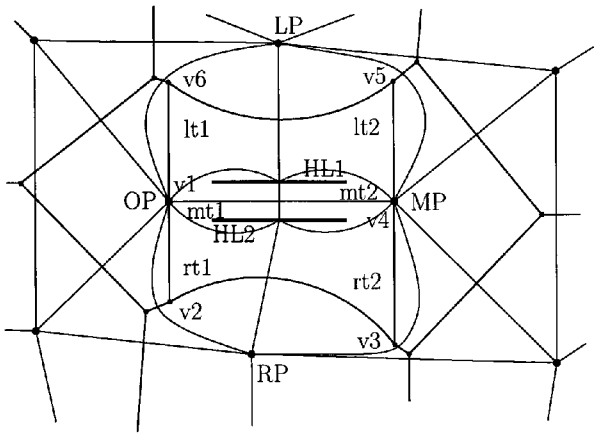


Figure 5.27: The Voronoi diagram after converting an edge to a line segment.

5.4.5 Convert a Line Segment into an Edge

Converting a line segment to an edge ("ConvertLineSegmentToEdge") is the inverse operation to converting an edge to a line segment (ConvertEdgeToLineSegment described in Section 5.4.4). It is performed as the final step of the line segment retraction operation, to remove a fully retracted line segment from the mesh by converting it to an ordinary edge and removing that edge together with two adjacent triangles.

The process can only be performed when the segment is enclosed by a quadrilateral containing inside seven edges and two half-lines, as in Figure 5.26b, so the configuration of the diagram surrounding the segment is the same as after it was created. It means that each of the half-lines together with their endpoints are connected to the same object on both sides of the segment. This can be tested by comparing if two edges originated at line segment endpoints' neighbouring edges overlapping half-lines (in Figure 5.26a: *e1* and *e3.Sym* on the left side and *e2* and *e4.Sym* on the right side) point at the same object for each side of the segment.

The first step of ConvertLineSegmentToEdge is a removal of the seven edges adjacent to the two half-lines of the line segment. They are disconnected using Splice and deleted. Then both half-lines are removed and a new edge connecting former endpoints of the segment is inserted. This forms two new triangles inside the quadrilateral for whom Voronoi vertices are calculated as the last step of the operation.

5.5 The Point Movement Procedure

In the moving point operation "MovePoint" a point *MP* is moved from the existing origin *OP* to the destination *DP* location (existing or not). Primarily the method has been developed in order to allow relocation of existing nodes of the mesh (Gold (1990)), but later its idea was found useful to insert line segments into Delaunay triangulations (Gold et al. (1997); Gold and Dakowicz (2006); Dakowicz and Gold (2006)), as *MP* can leave a trace behind as it moves, becoming a new line segment or a constrained edge. The relocation of existing nodes is an important part of the point insertion and deletion methods employed in this project, which in order to work with CDT and LSVD are different than those traditional VD/DT insertion (Guibas and Stolfi (1985)) and deletion (Devillers (1998)) algorithms. The insertion consists of two operations: splitting a new

node from an existing one and relocating it to the desired location. Similarly, the deletion consist of relocating the deleted node towards a selected node and merging them together. In this work, apart from relocating existing nodes and inserting line segments and constrained edges, MovePoint is used as well to retract line segments and constrained edges.

The point movement operation can be used to:

1. Move a point from *OP* to *DP* location. The moving point *MP* can be the origin point *OP* (this relocates *OP*) or a new point split from *OP* (this adds a new point).

2. Expand a constrained edge *CE* from *OP* to *DP*. *MP* is split from *OP*, the edge between them is marked as constrained (so *MP* becomes one of the endpoints of *CE*) and *MP* is moved to the location *DP*, expanding *CE*.

3. Expand a line segment *LS* from *OP* to *DP*. *MP* is split from *OP*, the edge between them is converted to a line segment *LS* (so *MP* becomes one of the endpoints of *LS*) and *MP* is moved to the location *DP*, expanding *LS*.

4. Retract a constrained edge *CE* from *OP* to *DP*. The moving endpoint *MP* can be split from *OP* (so *OP* remains in the mesh after the retraction) or *OP* can be *MP* (if *CE* is not connected to any other constrained edge this removes *CE* together with the endpoint *OP*). *MP* is moved to the location *DP*, retracting the constrained edge.

5. Retract a line segment *LS* from *OP* to *DP*. *MP* can be *OP* (so *LS* is retracted together with its endpoint *OP*) or a new endpoint of *LS* split from *OP*, as in the case of retracting constrained edges.

All those operations are performed using one procedure MovePoint, which consists of three main parts. In the first part the initial tests (including collision detection at the origin, destination and along line segments and constrained edges connected to *MP*) and operations (including Split if needed) are performed. Then the topological events are processed by testing intersections of the circumcircles of real and imaginary triangles adjacent to the moving point *MP* with the trajectory, as described in Section 4.5. In the third and last part *MP* is relocated to the location of the nearest topological event and an appropriate edge is swapped. Those two parts are repeated until there are no topological events detected along the trajectory and then *MP* is relocated to the destination location.

Although the MovePoint procedure is used for all possible moving point scenarios, some operations and tests are only performed for particular scenarios. The operations are distinguished by the byte "*operation*" argument. Depending on its value MovePoint performs different operations and produces different results:

- *operation* = 1 - move *OP* to *DP*, so *MP*=*OP* (*OP* relocated),
- *operation* = 2 - split *MP* from *OP* and move to *DP* (*OP* remains),
- *operation* = 3 - make a constrained edge from *OP* to *DP*, *MP* is split from *OP*,
- *operation* = 4 - make a line segment from *OP* to *DP*, *MP* is split from *OP*,
- *operation* = 5 - retract a constrained edge or line segment from *OP* to *DP* together with *OP*, so *MP*=*OP* (*OP* removed),

- *operation* = 6 - retract a constrained edge or line segment from *OP* to *DP* leaving *OP*, so *MP* is split from *OP* (*OP* remains).

Retraction of line segments and constrained edges requires an additional "*retractedObject*" parameter in the MovePoint procedure. The parameter is a quad-edge that identifies what kind of feature is being removed. When removing a constrained edge the *retractedObject* parameter is that edge directly, and when a line segment *LS* is being retracted then the destination of *retractedObject* (*retractedObject.Dest*) points to *LS*. This parameter is also used in Split (when retracting a constrained edge) and DisconnectLineSegment (when retracting a line segment) in operation number 6. In operations 1-4 the parameter *retractedObject* is set to nil.

The last parameter is a Boolean flag "*reconstruction*" , which is set to true in operations 3 and 4 when line segments or constrained edges are reconstructed after splitting them due to collisions. In all other cases this parameter is set to nil. When this flag is set to true the collision tests are not performed during the process, as the feature had already existed at that location before it was retracted.

Summarizing the above information the procedure call is MovePoint(*Origin Point, Destination Point, operation, retractedObject, reconstruction*).

The point *MP* can be a single point (when relocating a single node) or an endpoint of an existing constrained edge or line segment (when expanding or retracting them). The trajectory is a straight line defined by the *OP* and *DP* locations. However, the final trajectory can be modified into a sequence of lines (also non-straight) if *MP* collides with other objects.

If *MP* collides with a point *CLP* during the movement (as in Figure 5.28), then *MP* is relocated back to *OP* and then moved first towards *CLP*, and then (after merging them and splitting *MP* from *CLP*) from *CLP* towards *DP*, so the resulting trajectory is a non-straight line *OP-CLP-DP*.

If *MP* collides with a linear feature *L* (so it intersects the trajectory), then *MP* is relocated back to *OP* and *L* is split into two parts with a new point *CLP* at the collision location. Then *MP* is moved towards *CLP* and from *CLP* to *DP*, so the resulting trajectory is a straight line *OP-CLP-DP*.

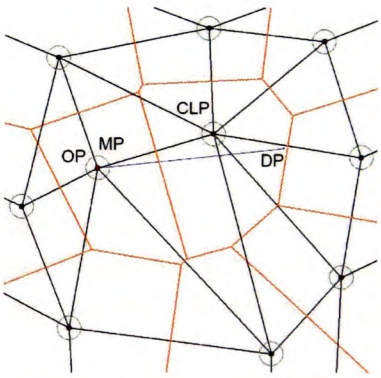


Figure 5.28: The trajectory modification from OP-DP to OP-CLP-DP.

After detecting a collision *MP* is always moved back to its initial location because merging colliding objects is usually not possible or produces overlaps with existing nodes. A colliding point CLP is usually not located on the trajectory and in the case of a line segment construction merging *MP* and *CLP* directly changes the segment direction, which often inducts other disks overlaps with the redirected segment. This is shown in Figure 5.29 where a new segment is being created from

OP towards *DP*. A collision is detected with CLP (Figure 5.29a) and *CLP* and *MP* are merged directly, which causes an overlap of a disk of another node c with the line segment (Figure 5.29). Moving back *MP* and directing it towards *CLP* avoids such situations and results in detecting a collision with c when moving MP towards CLP and eventually produces an *OP-c-CLP-DP* segments configuration, without any overlapping disks.

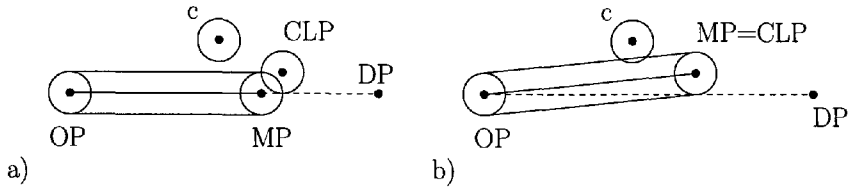


Figure 5.29: Collision of a line segment moving endpoint MP with a node. a) Detected collision with a node CLP. b) The diagram after merging MP and CLP directly, without shrinking the line segment first.

Also a collision with a linear feature *L*is handled by moving *MP*back first to the trajectory origin. A collision with *L*requires splitting *L*into two parts at the intersection point. However, the disk of *MP*often overlaps *L*(as it was moved towards it as far as possible) and at any static stage (when no kinetic operation is performed) there can not be overlapping disks left in the mesh, as described before in Section 5.3.2. Thus moving *MP*back to its initial location ensures the correctness of the graph at the moment of starting splitting *L*.

5.5.1 Point Movement and Expansion of Constrained Edges and Line Segments

This section is a brief description of the movement and expansion process. Additionally some examples of the process are presented. Please refer to Appendix B for a detailed description of this complex process.

In the first part of the process various initial tests are performed. The most important include collision tests at the *DP* location, tests if *DP* is not an endpoint of a line segment or constrained edge originated at *OP* or if there are line segments or constrained edges having another endpoint colliding with the trajectory. Those may lead to changes of the origin of the movement trajectory.

In the next step a new point *MP* is split from *OP* when adding a new point (operation 2), expanding a line segment or constrained edge (operations 3 and 4) or retracting a constrained edge (operation 5 with *retractedObject* being a constrained edge). When retracting a line segment (the operation 5 with *retractedObject* being a line segment) DisconnectLineSegment is called. This adds to the mesh one new point *MP* and two new zero-area triangles adjacent to the edge *SE* = *OP-MP*.

Then if requested, *SE* is converted to a constrained edge or a line segment. *SE* is made constrained by setting its *C* flag to true. Alternatively *SE* is converted to a line segment by the function ConvertEdgeToLineSegment, adding two half lines joining *OP* and *MP* and six new "triangles", as described in Section 5.4.4.

In the next step the real triangles located behind *MP* are identified by projecting their circumcentres (Voronoi vertices) onto the trajectory. These will be ignored in the movement process, as the real triangles are used for obtaining new neighbours for the moving point and these have Voronoi vertices in front of the moving point.

In the second main part of the process the potential topological events TE are tested to find the closest one and relocate the moving point to its position. The test trajectory is updated to *MP-DP* after each relocation of *MP*. The circumcircles of real and imaginary triangles (described in Section 4.5) are tested separately and they can intersects the trajectory, be tangent to the trajectory or not cross it. For the circles interacting with the trajectory the topological events are identified by testing positions of intersection points *CP*. A relative value of t is assigned for each location of *CP*, with values between 0 and 1 for locations between *MP* and *DP*, smaller than 0 for locations in front of *MP* and larger than 1 for locations behind *DP*. The circle intersecting the trajectory at the nearest position in front of *MP* (having the smallest t value) is selected as the one to be entered or left by *MP*.

While finding the nearest topological event collision tests are also performed. Collisions with nodes occur when CP overlaps a disk of any of the point neighbours of MP. Collisions with line segments or constrained edges occur when the trajectory of *MP*intersects them. The overlap of the disk of *MP*and the colliding feature is accepted and all topological events are processed until, in the CDT, *MP*is connected to both endpoints of a constrained edge crossing the trajectory or, in the LSVD, *MP*has the line segment crossing the trajectory as its only neighbour. Then no more movement is possible towards the colliding segment.

After testing all real and imaginary triangles the results are processed. If there was a collision with a linear feature detected then it has to be split in order to allow *MP* to reach the *DP* destination. In such a case *MP* is moved back to the origin position *OP* (which retracts the new line segment or constrained edge if necessary). Then if the intersection point location is closer than *2*DiskRadius* to any of the endpoints of the feature then the feature will not be split and that endpoint is the point where *MP* is going to be directed. Otherwise the feature is split at the intersection point location, which creates a new point *CLP* in the interior of the feature. Then *MP* is moved again, this time from *OP* towards *CLP*. After merging *MP* and *CLP* the movement is continued from the location of *CLP* towards *DP*.

If there is a collision detected with a node *CLP* then *MP* is moved back to *OP* (additionally the list *LE* of the "behind" triangles is reset), and then the whole process is started again from *OP* with *CLP* as the destination. This leads to merging *MP* and *CLP*. After that the process is initiated again by splitting *MP* from *CLP* and moving it towards the original destination *DP*.

If there was a topological event detected then *MP* is relocated to the position of the event. An appropriate edge is swapped, which in case of the real triangle leads to gaining a new neighbour, or in case of an imaginary triangle to losing one. Also the Voronoi diagram and the list of the behind triangles is updated.

Alternatively, if there were not any topological events detected between *MP* and the location of *DP* then *MP* is relocated to *DP*, just by changing its coordinates, and the Voronoi diagram is updated. If there was a vertex *P* already at the location *DP* then *MP* and *P* are merged together.

Figure 5.30 shows a process of moving a point from *OP* to *DP* location using MovePoint(*OP, DP, 1, nil, nil*) procedures call. The relocation involved two flips of edges - one to gain a neighbour (Figure 5.30b) and one to loose a neighbour (Figure 5.30c). *MP* is translated to *DP* (Figure 5.30d) when there are no more topological events detected between them.

Figure 5.31 shows a process of splitting a new point *MP* from an existing node *OP* and relocating it to the location *DP*. The procedure MovePoint(*OP, DP, 2, nil, nil*) call is used. After splitting *MP* from *OP* both nodes are at the same location (Figure 5.31b) and the movement is performed (Figure 5.31c-d) until there are no more topological events detected between *MP* and *DP* location,

so *MP* is relocated to *DP* (Figure 5.31e).

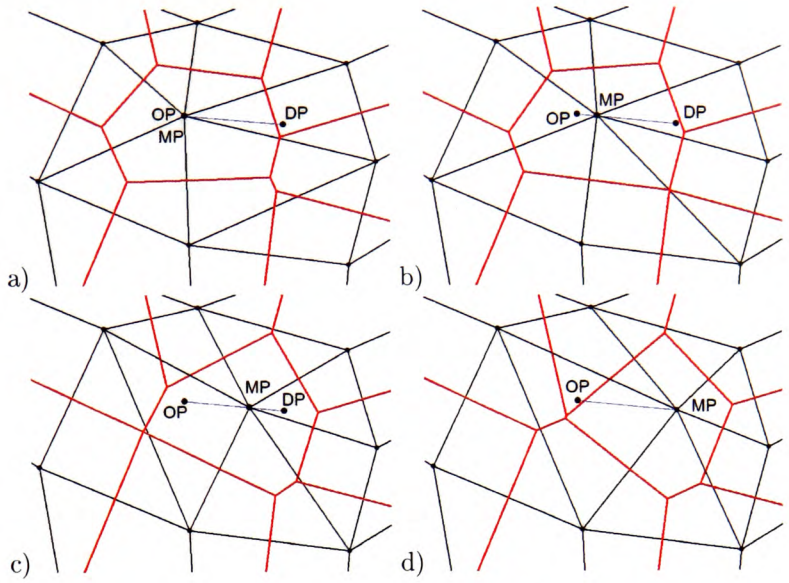


Figure 5.30: Moving a point MP from OP towards the DP location. a) The initial situation. b) After swapping one edge. c) After swapping the second edge. d) The final configuration.

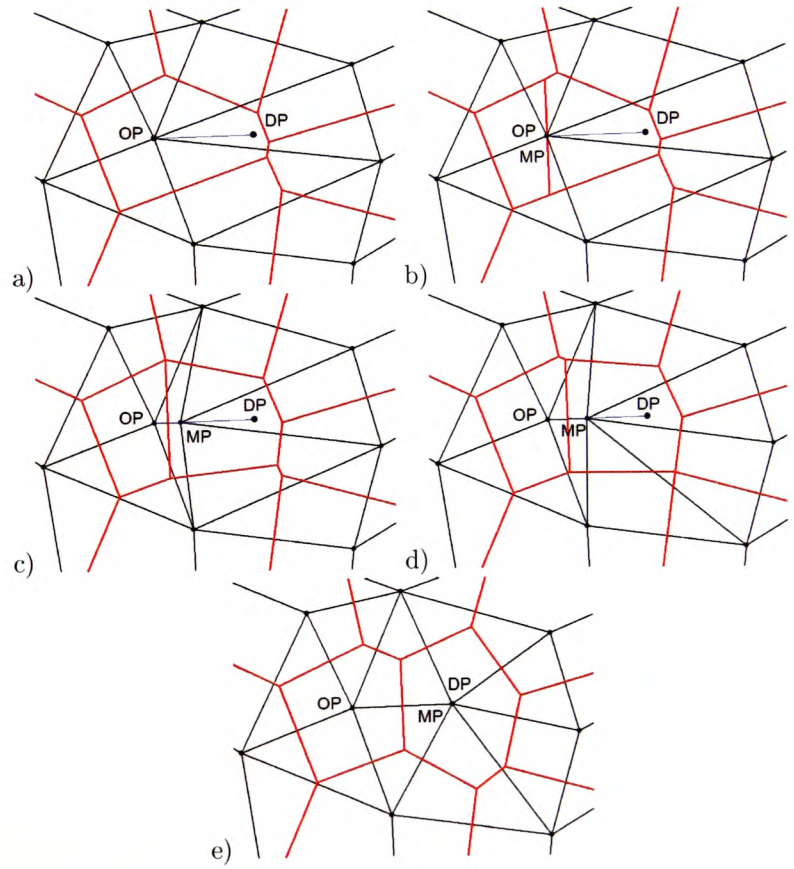


Figure 5.31: Moving a new point MP split from OP towards the DP location. a) The initial situation. b) After splitting MP from OP. c) After swapping one edge. d) After swapping the second edge. e) The final configuration.

Figure 5.32 shows a sequence of events leading to inserting a new constrained edge between *OP* and *DP* locations using MovePoint(*OP, DP, 3, nil, nil*) call. Firstly *MP* is split from the existing node *OP* and the edge between them is converted to a constrained edge (Figure 5.32b). Then the edge is expanded by moving its endpoint *MP* towards *DP* (Figure 5.32c-g) until there are no more topological events between them. Finally *MP* is relocated to *DP* (Figure 5.32h).

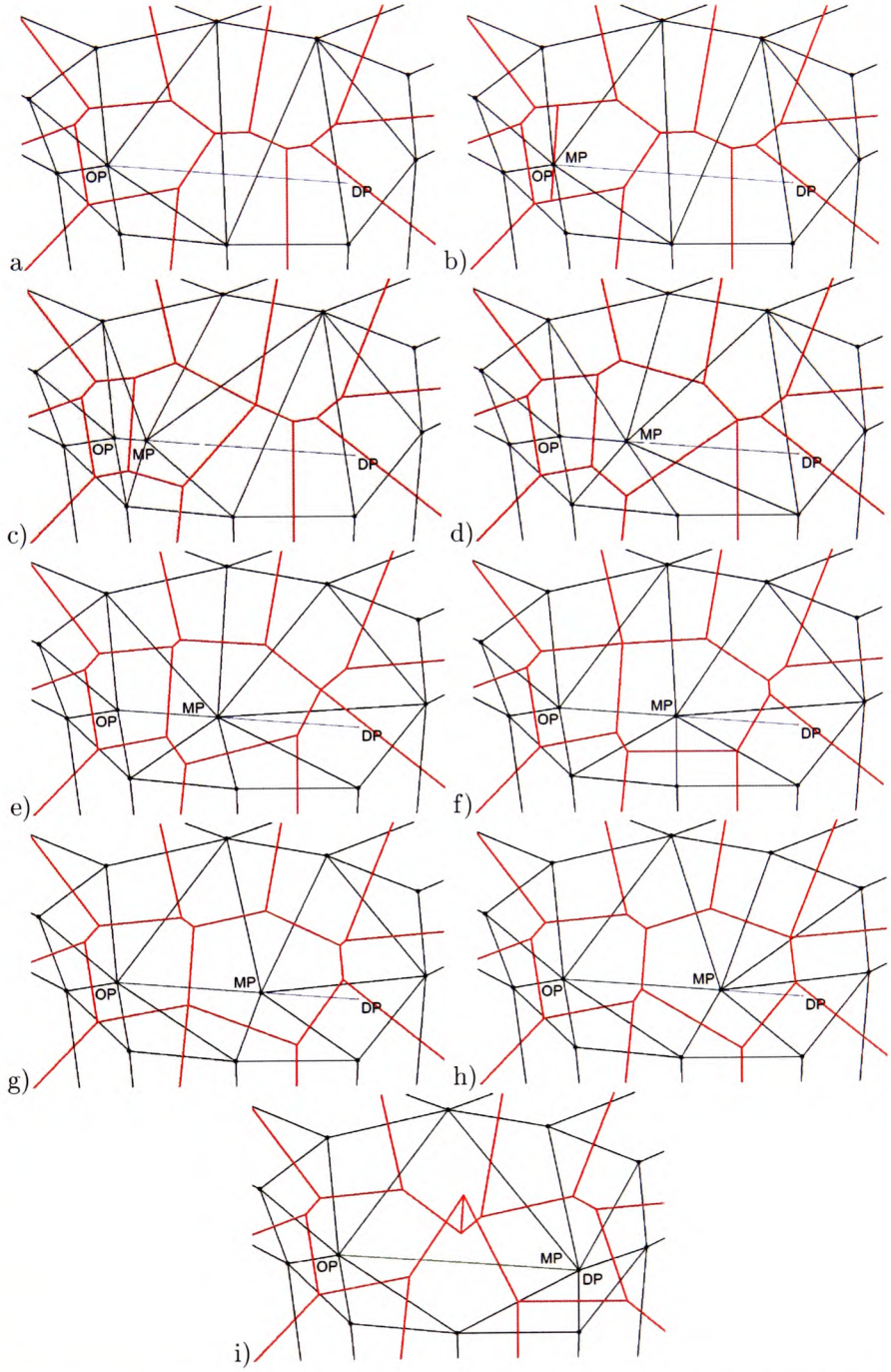


Figure 5.32: Making a constrained edge from OP to DP. a) The initial situation. b) After splitting MP from OP and converting the edge connecting OP and MP to a line segment. c-h) The expansion process. i) The resulting constrained edge.

Figure 5.33 presents insertion of a new line segment between locations *OP* and *DP* using

MovePoint(*OP, DP, 1, nil, nil*) call. Firstly, a new point *MP* is split from *OP* and the edge between them is converted into a new line segment (Figure 5.33b). The segment is expanded until no topological events are detected between the moving endpoint *MP* and *DP* (Figure 5.33c-d). In the last step *MP* is relocated to *DP* (Figure 5.33e).

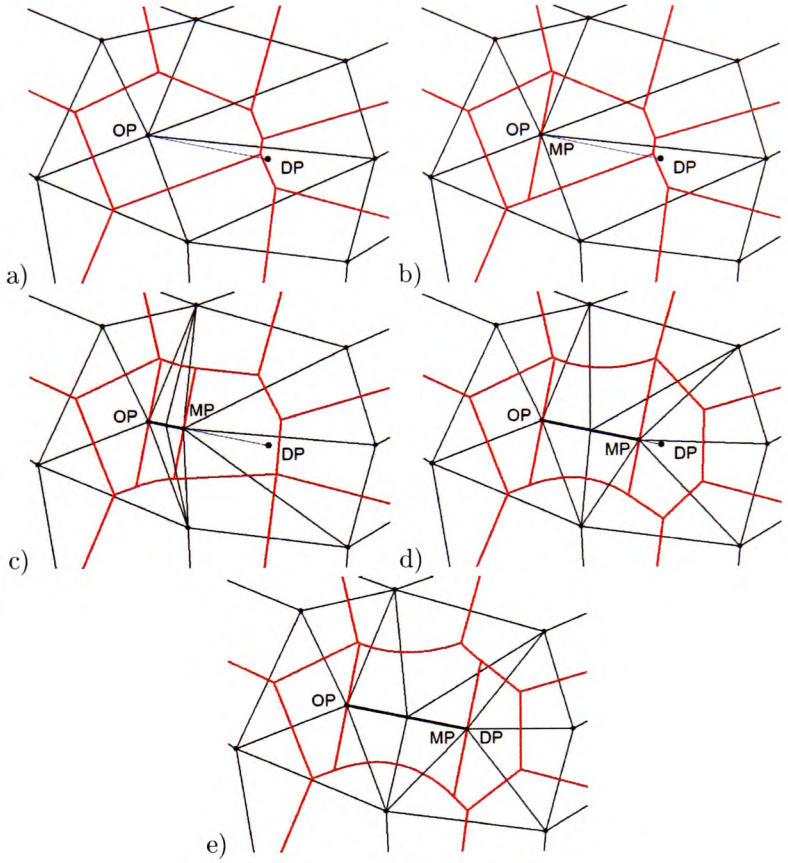


Figure 5.33: Making a new line segment from OP to the DP location. a) The initial situation. b) After splitting MP from OP and converting the edge connecting OP and MP to a line segment. c) After swapping one edge. d) After swapping the second edge. e) The final line segment.

5.5.2 Retraction of Constrained Edges and Line Segments

Removal of a line segment or a constrained edge is the inverse operation to their insertion. It is based on the same principles as the moving point process and it has been implemented in the same procedure MovePoint and called with the parameter "*operation*" equal to 4 or 5.

It is performed by retracting the feature from one of its endpoint towards the other by processing topological events along the trajectory. The process stops when no more movement is possible and the feature is removed after merging its endpoints (directly for a constrained edge or after converting a line segment into an edge).

The feature to be removed is a line segment or a constrained edge having *OP* and *DP* endpoints. It is identified using the parameter *retractedObject* of the procedure. When removing a constrained edges *retractedObject* is that edge directly, and when it is a line segment *LS* then the destination of *retractedObject* (*retractedObject.Dest*) points to *LS*. The point *OP* is the point from which the process starts, while *DP* is the destination of the movement.

In the first step the feature is disconnected from its *OP* endpoint if required (parameter *opera-*

tion = 6). For a constrained edge retraction the Split operation is used (described in Section 5.4.1), while retracting a line segment requires a special method, described in the Section 5.4.3. This adds a new point *MP* at the *OP* location, which becomes the new endpoint of the feature that is going to be moved towards *DP*. Also three new edges adding two new triangles to the triangulation are inserted. Points *MP* and *OP* are connected with a zero-length edge. In operation 5 this step is not performed and the endpoint *OP* becomes the moving point *MP*.

Then the real and imaginary triangles are tested to identify the nearest topological event location. However, when retracting a line segment not all edges are processed. As was mentioned before for line growing, the real triangles are processed in order to move *MP* forward by connecting *MP* with new objects located in front of it. There are only two edges that can be swapped. Those are originated on both sides of the line segment and the real circles are tangent to the segment (edges *et1* and *et2* and circles centered at *v1* and *v2* nodes marked in Figure 5.34a). After swapping one of them the line segment's neighbour becomes *MP's* neighbour. When processing the imaginary triangles objects are disconnected one by one from the endpoint *MP* of the segment. However, disconnecting an object should not add a new neighbour to the processed line segment. Only imaginary triangles that do not have the line segment as one of their vertices are processed. Such situation is shown in Figure 5.34b). The only two edges *e1* and *e2* considered for the swap are marked, together with the imaginary circles associated with their neighbours.

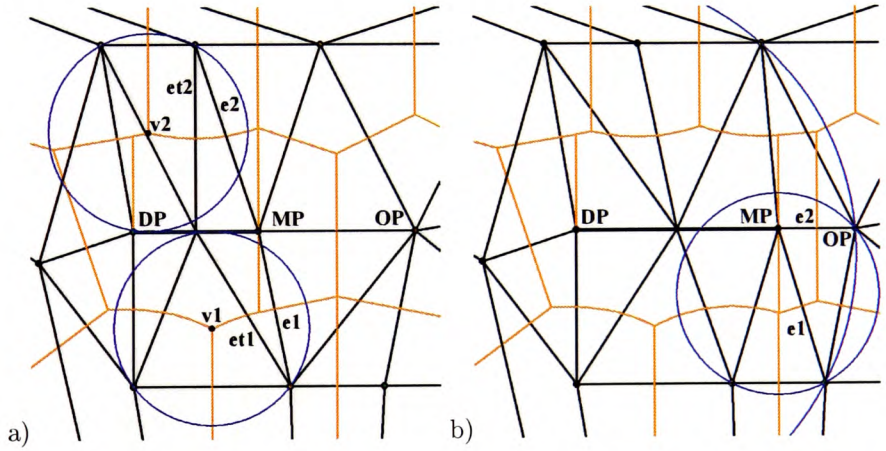


Figure 5.34: Retraction of a line segment. a) The real circles used in the process. b) The imaginary circles used in the process.

Potential topological events are tested and selected in the same way as in the moving point procedure for operations 1-4. The only difference is that the collision tests are not performed, as there are no collisions along the trajectory, which is the existing line segment or constrained edge. Topological events are processed by relocating the *MP* to the first topological event location and swapping an appropriate edge. This decreases the length of the feature.

If there were no topological events detected along the trajectory the feature can be removed from the mesh. In the case of retracting a line segment, it is converted to an ordinary edge using the method described in Section 5.4.5. Then the moving point *MP* and the destination *DP* are merged. This removes the edge between *MP* and *OP* and completes the retraction process.

Figure 5.35 shows each step of the process of retracting a line segment *LS*. The line segment has *OP* and *DP* endpoints which remain part of the structure after removing the line segment and the retraction is performed towards *DP*. Firstly the segment is disconnected from the *OP* endpoint.

This adds a new point *MP* being a new endpoint of *LS* (Figure 5.35a) and *OP* becomes a single point. *OP* and *MP* are at the same location and there is a zero-length edge between them, and two triangle adjacent to that edge (not seen in the figure, since *MP* and *OP* are at the same location). Then topological events are processed (Figure 5.35b-f) and the *MP* endpoint moves towards *DP* and appropriate edges are switched one by one. Processing the configuration in Figure 5.35f) does not detect any topological events between *MP* and *DP*, so the line segment is removed and *MP* and *DP* are merged, which ends the retraction process.

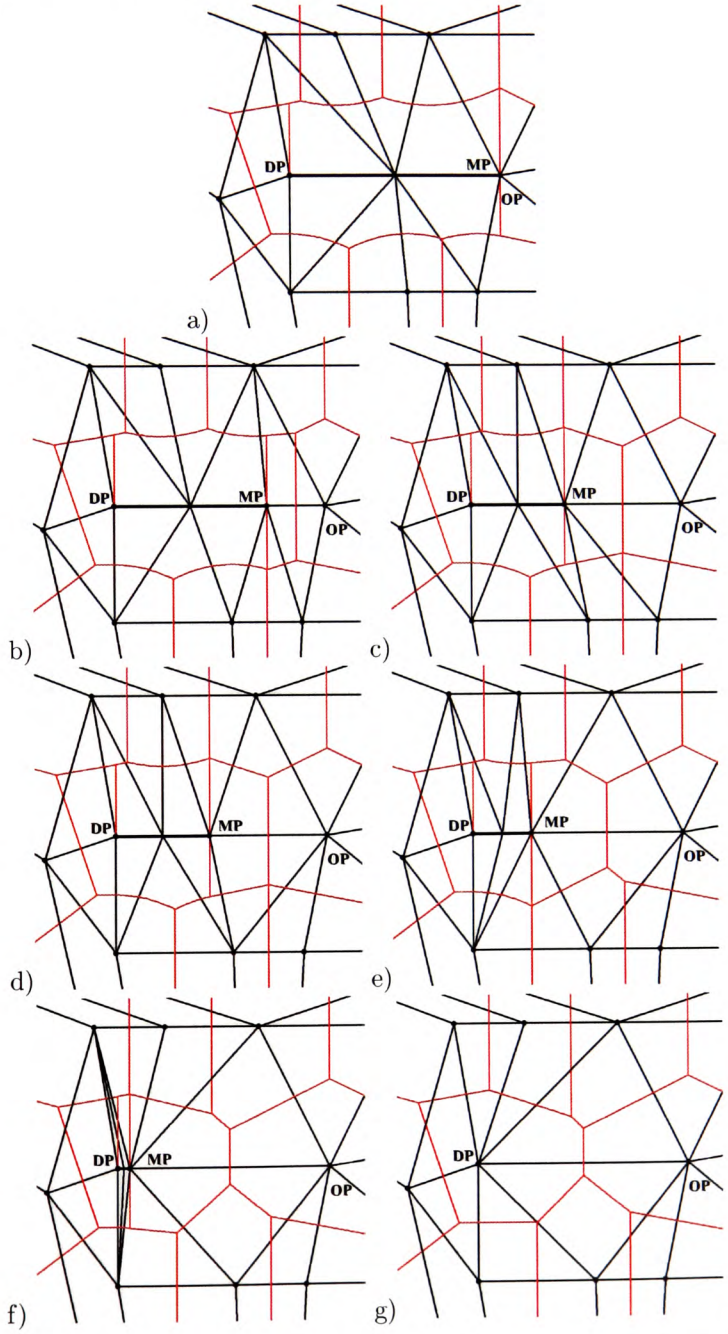


Figure 5.35: The consecutive steps of retracting a line segment between OP and DP, starting from OP towards DP.

5.6 High-level Operators

High-level operators are those directly available to the user. They allow construction and update of VD/DT/CDT/LSVD meshes. Some of them, such as move a point, make or remove a line segment or constrained edge are just calls of the procedure MovePoint with different parameters. Others, like insert or delete a point involve some processing, before calling MovePoint.

5.6.1 Move a Point

The operation of moving a point (Roos (1993)) is used to relocate an existing node from one location *OP* to the intended destination *DP*. It is performed using the MovePoint procedure with the operation parameter set to 1, so the Split operation is not used. The initial trajectory is a straight line connecting *OP* and *DP*, which can be modified during the movement if there are collisions with other nodes detected along the trajectory. If there are any line segment or constrained edges detected intersecting the trajectory, then the moving point is relocated back to *OP* as its relocation to *DP* is not possible. If there is already a node at the location *DP* then the operation Merge is called in MovePoint to merge them as the last step of the movement operation.

5.6.2 Insert a Point

The traditional incremental point insertion algorithm (Lawson (1972); Guibas and Stolfi (1985)) adds a new point to the Delaunay triangulation by locating the triangle *T*enclosing the insertion location (usually found using the Walk method), adding three new edges inside this triangle, and optimizing the triangulation for triangles adjacent to *T*to fulfill the Delaunay empty circumcircle property. In the CDT the incremental algorithm can be used, with one modification: that constrained edges are fixed and cannot be swapped. In the LSVD the point insertion cannot be performed using this simple method as edges connecting points and line segments are not straight lines but just links connecting Voronoi neighbours, and algorithms based on the orientation tests (requiring three points) cannot be used. Since there are no "straight edge triangles" it is not obvious which triple of objects to choose as the "triangle" enclosing the insertion location (which would need to be divided into three new triangles with the new point inside).

Since the incremental algorithm cannot be used in the LSVD, the idea used in this project is to temporarily insert a new point by splitting it from any existing vertex and move it towards the intended final insertion location. This method works with any type of triangulation.

The insertion is attempted at some location *DP* with x, y coordinates. In the first step the nearest object to *DP* is located using Walk. Walk returns an edge e with the origin $e.Org$ being the nearest object to *DP* (*DP* is inside the Voronoi region of $e.Org$). There are three different options in the algorithm depending what object is returned as $e.Org$. It can be one of the big triangle vertices, an ordinary point or a line segment object.

If both endpoints of e are vertices of the big triangle it means that this is the insertion of the first point in the mesh (if only one of them is a vertex of the big triangle then the other endpoint can be used). Such a case is handled separately, as splitting a new point from the big triangle vertices is not possible. In this case three new edges connecting the new point and the big triangle vertices are created using the MakeEdge and Splice functions. After inserting the first point the next new point can be split from the first one.

Another case is *e.Org* being an ordinary point of the mesh. In such case a new point *NP* is split from *e.Org* and moved towards the *DP* location.

The most complex case is when *e.Org* is a line segment, or more precisely a half-line *hl* of the line segment that is the closest object to *DP*. Then the collision tests are run to test if *DP* collides with *hl*'s interior or one of the endpoints. It is done by testing if the distance from *DP* to *hl* is not smaller than *2*DiskRadius* and if so then *DP* is projected onto *hl* and the distance from the projection location *px,py* to each of the endpoints of *hl* is computed. If *x, y* collides with any of the endpoints then the insertion at that location cannot be performed. The colliding endpoint of *hl* is passed as the result and the function terminates.

If *x, y* collides with the interior of *hl* then *hl* has to be split into two segments at the location of the projection *px, py* of *DP* onto *hl* and their common point is the result of the insertion. Firstly the endpoint closer to *px, py* is selected (to minimize to distance of the line retraction, *LP2* in Figure 5.36a) and then *hl* is retracted towards *px, py* by moving *LP2* (the original point *LP2* stays at its location and a new endpoint of *hl* is split from *LP2*). After reaching px, py a new point *NLP* is split from the current location of *LP = (px, py)* and a new segment is constructed by moving *NLP* towards the original endpoint *LP2*. *NLP* and *LP2* are merged at the end and as the result the new point is inserted on the line segment between *LP1* and *LP2* endpoints (Figure 5.36b).

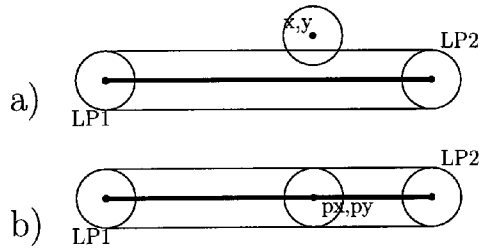


Figure 5.36: Insertion of a point colliding with a line segment. a) The location x,y collides with the segment. b) After splitting the segment.

If the origin *e.Org* of the nearest object to the location *DP* is a half-line *hl* and it does not collide with *x, y* then firstly a point from which the new point is going to be split has to be found. This also should be the nearest point to *DP* to minimize the amount of movement from that point to the location *x, y*. This is done by testing the distance from *DP* to each of the points connected to *hl*. The selected nearest point *NP* can be one of the endpoints of *hl* or any other neighbouring point, as in Figure 5.37a. In the next step a new point is split from *NP* and moved towards the location *DP*, as in Figure 5.37b.

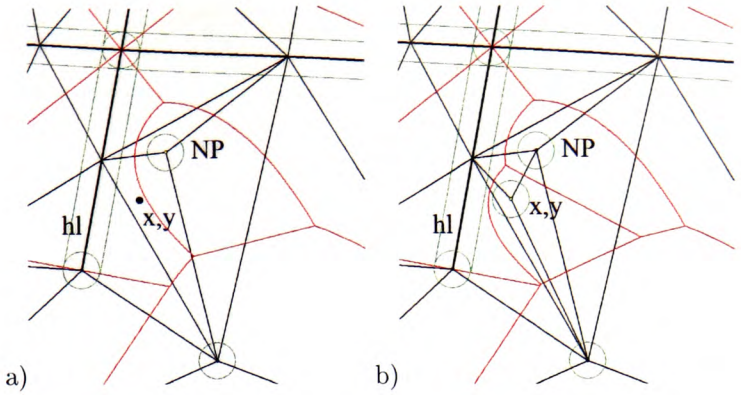


Figure 5.37: Insertion of a point at the x, y location inside the Voronoi cell of a line segment hl. a) The initial diagram. b) The result

5.6.3 Delete a Point

Deletion of a point in the DT/VD has been studied extensively and several algorithms exist (Mostafavi et al. (2003); Devillers (1998)). For the CDT, Shewchuk (2003) proposed a method of updating (this involves points removal) and constructing the diagram using bistellar flips. Kallmann et al. (2003) mentioned a CDT point deletion while describing removal of constrained edges and suggested deleting edges adjacent to the removed point and retriangulating the resulting polygon. For the LSVD Gold (1990); Yang (1997) proposed moving the point towards another node and removing it by merging them. This method is applied in this research and is used also for the CDT and the VD/DT meshes.

In this project, removal of a point utilizes the moving point mechanism and the collision handling technique by merging colliding nodes. The known "ear" methods are not used, as they require "straight edge triangles" for geometric tests, not available in the LSVD. The movement is performed using the method described in Section 5.5, while the merging operation is described in Section 5.4.2. The same operation is used for all types of mesh.

A single point P (not being an endpoint of a constrained edge or a line segment) is removed from the mesh by moving it towards an arbitrary selected point AP. This is performed by changing its location and swapping appropriate edges, until no further movement is possible. Then P and AP are merged together, which removes P from the triangulation. The selection of the point AP that is the destination of the movement process and will absorb point P can be done in various ways and the method used here is to use the nearest point connected to P. This can be a single point or an endpoint of a neighbouring line segment or constrained edge.

If a point P is connected to one or more constrained edges then its removal involves more operations, as all constrained edges having P as one of their vertices have to be removed (edges *ce1* and *ce2* in Figure 5.38a). Removal of a constrained edge *ce* connecting P and another point AP is performed by disconnecting *ce* from P, which creates a new endpoint NP, and moving NP towards AP (see Section 5.6.5). This retracts the constrained edge *ce*, until NP and AP are merged. After removing all adjoining constrained edges one by one (Figure 5.38b and 5.38c), point P becomes a single point and is removed by moving it towards one of its neighbouring points and merging them together (Figure 5.38d).

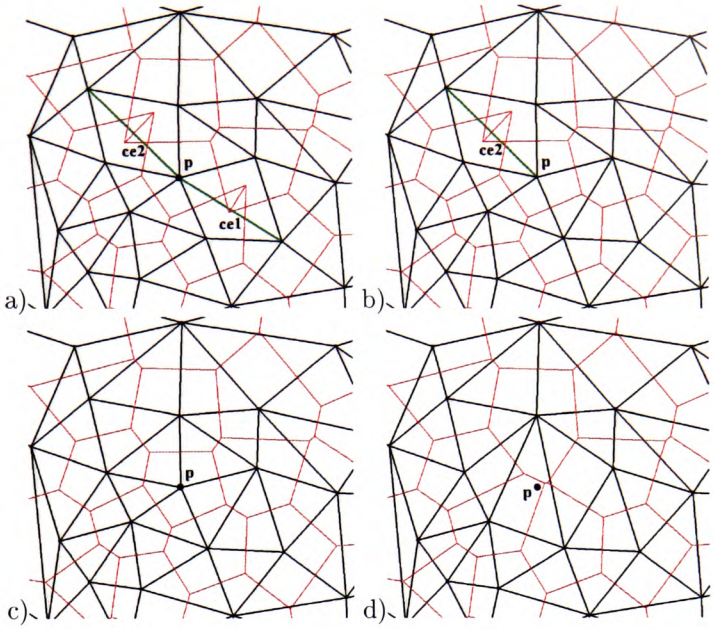


Figure 5.38: Deletion of a common endpoint p of two constrained edges ce1 and ce2. a) The initial situation. b) After deletion of ce1. c) After deletion of ce2. d) After deletion of p.

A similar situation is the case of removing a point being an endpoint of one or more line segments in the LSVD, as shown in Figure 5.39a. Here two adjoining line segments are removed (see Section 5.6.5) by retracting each of them towards one of their endpoints (Figure 5.39b and 5.39c) leaving their endpoints in the mesh. Then the remaining common endpoint of two segments is deleted by moving it towards one of the neighbours and merging them together (Figure 5.39d).

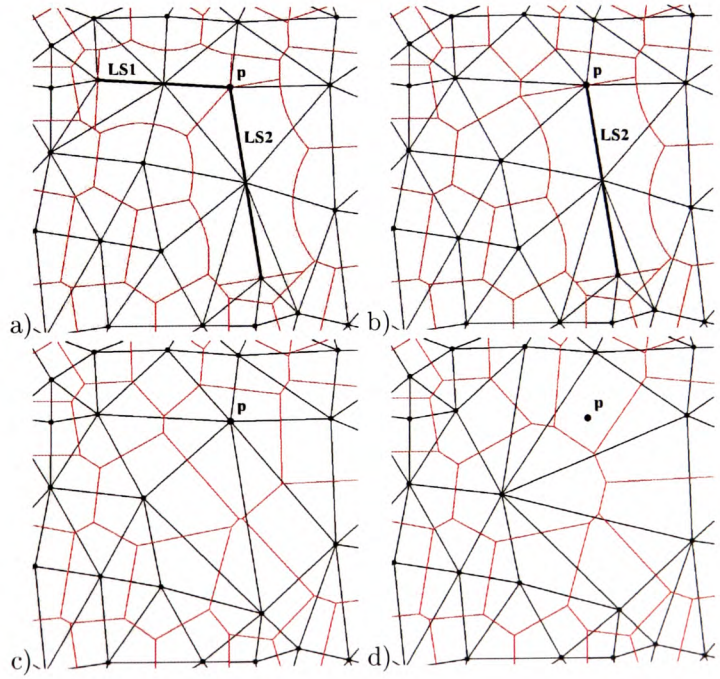


Figure 5.39: Deletion of a common endpoint p of two line segments LS1 and LS2. a) The initial situation. b) After deletion LS1. c) After deletion of LS2. d) After deletion of p.

5.6.4 Insert a Constrained Edge or Line Segment

The insertion of a constrained edge or a line segment feature into the mesh is performed using the MovePoint operation. MovePoint is called with the locations *OP* (existing in the mesh) and *DP* (existing or not) for the endpoints of the feature to be created and the *operation* parameter set to 3 when constructing a constrained edge or set to 4 when constructing a line segment. One endpoint *OP* of the feature already exists in the mesh and the other endpoint *MP* is split from *OP* in the first part of the MovePoint procedure. Then the edge between *OP* and *MP* is converted to a constrained edge (setting its constrained flag *C* to true) or a line segment (calling the ConvertEdgeToLineSegment procedure). Then the feature is expanded towards the location of *DP* by relocating the endpoint *MP*. If there are no collisions during the expansions then the resulting feature is a straight constrained edge or line segment connecting *OP* and *MP=DP*. If there were collisions then the resulting object can consist of several connected line segments or constrained edges. If there was a node at the location *DP* then *MP* is merged with that node in the last step of the expansion operation. Figure 5.40a shows the mesh with two nodes *OP* and *DP* marked and Figure 5.40b shows the resulting mesh after inserting a line segment connecting *OP* and *DP* nodes.

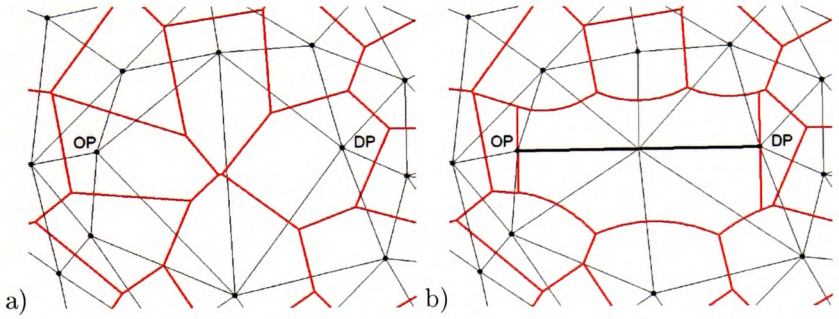


Figure 5.40: Insertion of a line segment. a) The initial mesh. b) After inserting a line segment connecting nodes OP and DP

5.6.5 Remove a Constrained Edge or Line Segment

Removal of a constrained edge or a line segment is the inverse operation to the insertion. It is performed using the same MovePoint procedure with two endpoints *OP* and *DP* of the feature passed as the arguments and *DP* is the endpoint towards which the retraction will be performed. The parameter "operation" is set to 5 when the endpoint *OP* is going to be the moving endpoint *MP* (*OP* is removed from the mesh), or to 6 when *MP* is split from *OP* (*OP* remains in the mesh). The parameter "*retractedObject*" which is a quad-edge identifies the feature that is being retracted. When removing a constrained edge the parameter *retractedObject* identifies that edge, and when a line segment is removed it is identified by the destination object of *retractedObject* (*retractedObject.Dest*). The feature is retracted until no topological events are detected between *MP* and *DP* and is removed merging endpoints *MP* and *DP* (directly when removing a constrained edge or after converting the line segment between them to an edge).

5.7 Robustness

Computation of the LSVD is a very complex issue and this is the reason why there are many more theoretical studies of the problem than its actual implementations (Held (2001)). Computing the point based VD can be reduced to assessing the sign of a determinant (the orientation test) and based on it InCircle test, but the LSVD construction goes beyond that. As noted by Held (2001) the difficulty of obtaining robust algorithms for the LSVD construction is given by the fact that "the computations required during the construction of the Voronoi diagram of polygonal data go well beyond the determination of signs of determinants, and also well beyond the domain of rational arithmetic".

Burnikel et al. (1994) discussed robustness issues of three different approaches to constructing the VD/LSVD. One uses floating point arithmetic usually employing tolerance values to help making decisions. However, the inexact versions of the geometric predicates are vulnerable to wrong answers caused by roundoff errors and this can lead geometric algorithms to produce incorrect input or crash (Shewchuk (1996)). The other method uses floating point arithmetic as well, but resolves the output of "doubtful" tests arbitrarily - consistently with previous tests. The third approach uses exact arithmetic which guarantees that "all numerical values are represented exactly and all branching decisions are error-free" (Yap and Dubé (1995)). This assures robustness of results, although often at the cost of slowing down the speed of the program by a factor of ten or more. However, there are various techniques improving the speed of exact computations, including the adaptive method of Shewchuk (1997), where the result is only computed to the precision necessary to guarantee correctness. Still, exact arithmetic is expensive and it should be avoided when possible (Shewchuk (1997)). Also in algorithms that construct new geometric objects, such as circumcircles or segment intersections, exact arithmetic is "constrained by its cost and its inability to represent arbitrary irrational numbers" Shewchuk (1997).

It appears that the only robust incremental implementation of the LSVD construction, besides the very complex method developed by Held (2001) (based on floating point arithmetic but requiring a very careful implementation of the numerical computations, an automatic relaxation of tolerance thresholds and a multi-level recovery process with "desperate mode"), is the work described in Karavelas (2004) and currently included in the CGAL library (Karavelas (2006)). It handles degenerate cases via a perturbation scheme and relies on a special representation of line segments, special arithmetic filtering (to quickly filter out the easy cases) and geometric filtering (to eliminate situations in which the arithmetic filter can fail) techniques used in all predicate evaluations and employment of the Exact Geometric Computation paradigm formulated by Yap and Dubé (1995). This paradigm is used throughout in CGAL and it states that "robustness of the algorithm is ensured if and only if the predicates are evaluated exactly, since the branches in the algorithms are based on the results of predicates" (Alliez et al. (in press)). This method uses special a representation of intersection points to handle intersecting segments, which exploits the configuration and construction history of the geometric data. This makes the method robust, but at the same time does not allow deletion of nodes from the diagram.

Our approach allows insertion and deletion of points, constrained edges and line segments, unlike the methods of Held (2001) and Karavelas (2004), which allow the insertion of sites only. The implementation uses floating point arithmetic because to our knowledge there is no exact method available for calculating circumcentres for line segment objects and it appears that none of the papers addresses the issue of calculating the circumcircle in the LSVD. The iterative circumcircle

calculation method (Anton and Gold (1997)) employed in this work produces approximate results only, so the method has to deal with inaccurate locations of Voronoi vertices and using exact arithmetic would not improve its robustness much. Also the priority was to have a fully dynamic model allowing insertion and deletion of nodes, so using the listed above approaches (not supporting deletion) was not possible.

Given the problems of floating point arithmetic precision the key issue is the robustness of the method. "Any arithmetic operation not resulting in a topological change causes no robustness problems" (Gold and Dakowicz (2006)), so robustness problems can be mainly caused by geometric tests and operations used to trigger topological changes. Using floating point arithmetic requires employing tolerance values to aid decision making processes. The tolerance sizes are dependant on the numerical size of coordinates and should be selected carefully.

Our kinetic method is driven by the processing of topological events along the moving point trajectory. Locations of topological events are detected by calculating the intersections of real and imaginary circles with the trajectory. The circles are computed using the iterative method, which calculates the circle for three input objects iteratively until two consecutive circumcentres are within a specified tolerance value. This means that the resulting circumcentre of three objects (unless three points are used as input objects) is always approximate. The method uses a prespecified stopping tolerance value to terminate the process when two consecutive circumcentres are within this value. When it is too large the resulting circumcentre might be in a relatively large distance from its exact location, which may cause problems later when it is projected onto segments and falls outside them instead of inside. When the value is too small it can never be reached as the circumcentre location can start "oscillating".

As noted in Held (2001) "numerical problems caused by inaccuracies of the floating point arithmetic may manifest themselves later on during the execution of the algorithm". Inaccurate positions of resulting Voronoi vertices in many cases might never cause problems, but problematic situations can occur if these vertices are used to make decisions about changes in topology. This happens for example when positions of their projections onto the moving point trajectory decide about selection of edges to be duplicated in Split, or about rejecting or accepting the nearest topological event when expanding a line segment. A tolerance value is used to snap the projection results to the endpoints of linear features (the trajectory or line segments) when projecting Voronoi vertices onto them. Another tolerance used is the *DiskRadius* value itself, which defines buffer zones of objects and is used to detect collisions of the moving point with other objects.

In Split the projections of Voronoi vertices (all Voronoi vertices are approximate) of the current Voronoi cell onto the trajectory are used to determine two Delaunay edges that are going to be "duplicated". This is a critical decision, as selection of wrong edges when constructing a new line segment can lead to the wrong configuration of the initial diagram, so the line segment expansion process can not be performed correctly and the final diagram is not correct. Such problems may happen when there are several Voronoi vertices at the same position (several cocircular objects) that are "perpendicular" to the split location and their projections fall almost randomly in front or behind it, instead of directly onto it, so the selection process is problematic. This is a weak point of the algorithm and should be improved in future work.

When finding the next topological event for the moving point, intersections of the trajectory with circumcircles are imprecise as the circumcircles themselves are imprecise. A tolerance is used to check if the intersection point is not too close to the trajectory origin or destination, and is snapped to their locations if necessary. This permits processing all topological events, even these

falling behind the destination point due to arithmetic inaccuracies.

Problems can also occur when working with distances that are smaller or similar to the tolerance values. This happens for example when intersecting connected at a very small angle two very long line segments with another segment near their common endpoint. The intersection points calculated using float-point arithmetic for such configurations may be located in relatively large distances from their exact positions (not on line segments), in extreme cases even on the other side of another segment, leading to invalid diagrams.

Detecting if three points are collinear is problematic when using floating point arithmetic. Adding a new node onto the existing line segments (splitting a segment into two parts) ideally should results in three collinear points, but due to floating point inaccuracies this seldom happens and a very large circumcircle is associated with these points. Such cases can be detected using a tolerance test in their determinant value or by testing if the potential circle is not too large for the input map, although this raises another issue: how large circles are too large?

Another important issue is to guarantee the same result of geometric tests for the same set of objects, especially in determinant and circumcircle calculations, as a different order of input objects in geometric calculations gives different results due to computers arithmetic imprecision. To ensure the consistency of results, all objects are sorted lexicographically, according to x and y coordinates and passed in this consistent order to subroutines.

In future work some kind of numerical "sanity tests", similar to these described in Held (2001) could be applied at chosen stages of the process to monitor the correctness of the VD. Also, if possible, it would be worthwhile to research a possibility of implementing the whole algorithm using exact arithmetic with exact calculations of the circumcircle, instead of the iterative method. Nevertheless, as demonstrated in many of the figures, our algorithm is effective in the vast majority of cases.

5.8 Summary

As described in this chapter, the algorithms and special cases of CDT and LSVD construction are complex. All workers have reported this. Our approach was designed to be kinetic, allowing construction of maps from any type of point and line input, and dynamic map updating by insertion and deletion of points and lines. The method took considerable effort to design and implement and can be used for practical applications. With this we can achieve our objective of a common data structure and set of operators for the major classes of GIS data.

While not particularly fast our interactive algorithm runs in linear time. Movement of a point MP, as well as insertion and deletion of a constrained edge or a line segment having MP as the moving endpoint, requires testing all neighbours of MP to find the nearest topological event and this is repeated until the destination is reached. The number of iterations depends on the complexity of the diagram in the vicinity of the trajectory. Any collision detected during the movement adds a number of operations to the process and requires moving MP back to the origin and then moving it towards the collision location (see Section 5.5). Additionally every collision with a line segment or a constrained edge requires splitting it into two parts at the intersection point with the trajectory. This requires using the moving point procedure twice - to shrink the line to the intersection point location and then to rebuild its original shape from there.

Memory requirements are the same as for the quad-edge structure of Guibas and Stolfi (1985) which simultaneously store the Delaunay and Voronoi graphs. In the CDT, in order to distinguish

between ordinary and constrained edges, an additional "constrained" flag is added to each TQuad of the quad-edge, while in the LSVD the half-line pairs with additional triangulation edges (as described in Section 5.4.4) are needed for storing line segments.

If in the long term our dynamic approach can not be made fully robust, nevertheless the unified spatial data structure for GIS remains valid and the LSVD in such a case can be constructed using the robust methods of Held (2001) or Karavelas (2004), although losing the ability to modify the map by deletion of points and line segments.

Chapter 6

Applications

This chapter discusses the realization of typical GIS operations using the Voronoi diagram, Delaunay triangulation and their variations. The chapter starts with a discussion of differences between the CDT and LSVD based models (Section 6.1.1) which shows that the LSVD is a better-specified model of the spatial relationships for map objects built from points and line segments than is the CDT.

Then examples of the four categories of spatial data types are presented in Sections 6.1.2- 6.1.5. Discrete objects, networks, polygons and surfaces (introduced in Chapter 1) are all represented by Voronoi diagrams (the ordinary or line segment) or Delaunay triangulations (the ordinary or constrained) having the same quad-edge based structure.

Then various applications are listed for each of the four categories. Section 6.2 presents operations and applications for discrete objects, including geological mapping, rapid digitizing and Marine GIS. Operations on Voronoi based networks are described in Section 6.3, including traditional network analysis and watershed generation. Section 6.4 presents some traditional GIS operations performed for polygonal maps, including reclassification and dissolving. Finally, various operations possible for Voronoi based surfaces are described in Section 6.5, including generation of good quality terrain models, interpolation, terrain modification and runoff modelling.

The last part of this chapter deals with GIS operations involving more than one layer. The two traditional operations of this type are "point in polygon" and "polygon overlay" (called in this research "query" and "merge" respectively). They are introduced and described with examples in Section 6.6.

6.1 Spatial Data Categories Represented as Fields

This section starts with the discussion of differences between the LSVD and the CDT based models and shows why the LSVD is more useful in spatial analysis than the CDT. Then examples of four different categories of spatial data types represented by Voronoi diagrams are provided. Simple discrete objects such as points are converted to fields by calculating the ordinary Voronoi diagram, which structure allows more types of analyses than unconnected set of points. Polygonal objects such as houses can be converted to fields by calculating a LSVD or CDT from segments defining polygons, which makes proximity to a location a trivial issue (similarly for linear objects). Additionally mobile objects can be easily handled using the kinetic Voronoi diagram. Another category is space partitioning polygons. Representing them with the CDT/LSVD eliminates many

problems of map construction. The problem of creating gaps and overlaps between adjacent polygons (shivers), as well as overshoots and undershoots where joints are not exact, is greatly reduced by the collision handling mechanism used in the MovePoint procedure (fixed size zones associated with objects and the moving endpoint of the line segment). All networks can be converted to fields by constructing the LSVD or CDT from their segments, while those densely sampled by constructing the ordinary point based VD and extracting the crust also. Flow can be easily simulated is those diagrams, and such representations add possibilities of additional analyses as well, such as the proximity to the nearest network segment in the LSVD. When constructing networks the overshoots and undershoots can be prevented using the MovePoint technique, in the same way as when constructing polygons. Finally all types of surface models (which are fields represented by vectors or rasters) can be converted to the DT by triangulating data points, contour lines or grid cells. Resulting TINs, together with extracted VDs, can then be analysed in the same way no matter what their original format was.

6.1.1 CDT vs LSVD

The ordinary point based VD can not be used to convert data sets with linear features (representing separate linear features, networks or polygons) into fields as it does not guarantee preserving connectivity of line endpoints, and additional nodes along lines are often required to preserve their shape. Thus to preserve connectivity of endpoints of linear objects without modifying the input data the LSVD or the CDT should be constructed from such data. Each line is present in the LSVD as a line segment object, while in the CDT as a constrained edge. Strings of lines are represented as connected sets of line segment objects or constrained edges respectively. However, these two diagrams possess very different properties.

Figure 6.1 presents two different representations of the same polygonal map of buildings. In Figure 6.1a the map is converted to the line segment VD and the buildings are represented using polygons formed by closed loops of line segment objects connected at common endpoints. All endpoints and line segments have their own Voronoi cells associated with them. Voronoi cells of nodes and line segments defining each polygon form together Voronoi regions of polygons, defining their proximal regions and clearly establishing neighbourhood relationships between polygons (Gold (2006)). Figure 6.1a shows a building *ba* with its Voronoi region darkly shaded. This region shares Voronoi edges with Voronoi regions of seven buildings *bg-bn*, which are defined in the diagram as neighbours of *ba*. Additionally the interior of each building is clearly defined (for example the lightly shaded cells of *ba* in Figure 6.1a) and the medial axis is available.

Figure 6.1b shows the same set of polygons converted to the constrained DT, where boundaries of polygons are represented by constrained edges. Polygons here are not objects like in the LSVD, but just sequences of constrained edges, that differ from the ordinary Delaunay edges only by their "constrained" attribute. Neighbourhood relationships between buildings are established by Delaunay edges connecting endpoints of constrained edges, while in the LSVD both types of objects - the line segments objects themselves and their endpoints, have neighbours. In the CDT the 2D Voronoi diagram is generated from points using the visibility property (Edelsbrunner (2000)) and some Voronoi cells are "not complete" because they are "cut" by constrained edges. Some Voronoi edges are completely excluded from the diagram, because both their Voronoi vertices are invisible from their generators, which can be seen in Figure 6.1b where the dual Voronoi edge of the constrained edge *ab*is not present in the VD as its endpoints (circumcentres of *abc*and *aeb*) are

not visible for generators c and e on opposite sides of ab, and the dual of ac is not present either. Some dual Voronoi edges are only partly present, as one of their endpoints is invisible from an endpoint of its Delaunay edge (for example the dual edge of bc in Figure 6.1b marked with a dashed line). Thus, unlike in the LSVD, the Voronoi cells in the CDT often do not define proximal regions of polygons formed by constrained edges. This can be seen in Figure 6.1b where the Voronoi boundary between polygons bn and bo is not half-way between constrained edges nm and op (as it is in the LSVD in Figure 6.1a).

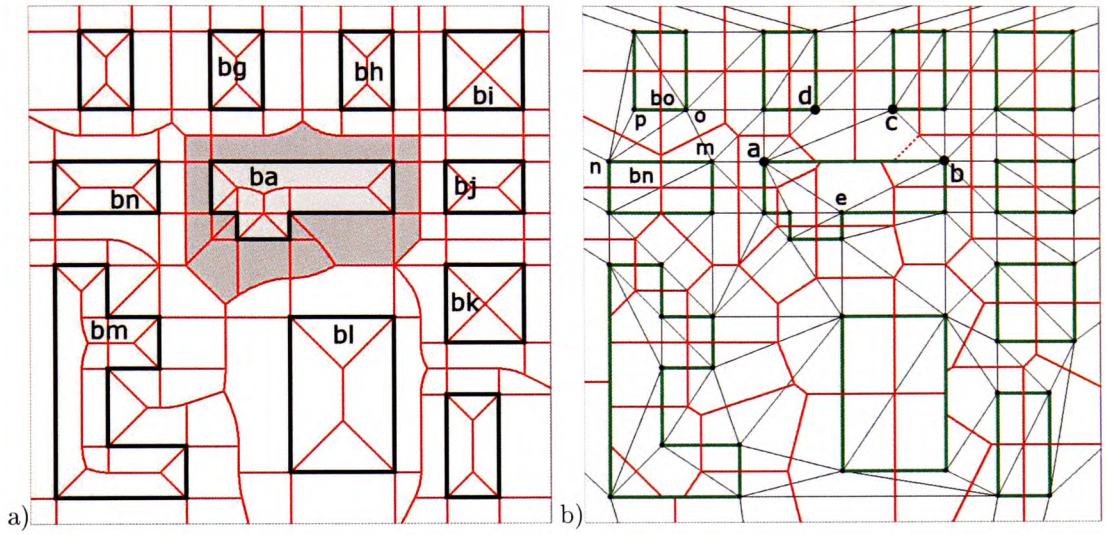


Figure 6.1: Polygonal discrete objects. a) Represented by the LSVD. b) Represented by the CDT.

Line segments in the LSVD and constrained edges in the CDT can be used for storing attributes, but in a different way. When representing common boundaries shared by two objects, such as adjacent polygons representing postcode boundaries or land ownership, the two-halflines representation of line segments in the LSVD becomes very useful, as each of the two adjacent half-lines can have its own attribute referring to a different polygon. This is different in the CDT, where each part of the boundary is represented by a constrained edge whose structure is the same as the structure of an ordinary triangulation edge, only with a different is a value of the "constrained" flag. Using the quad-edge structure for storing edges of the mesh (see Section 4.1) adds the possibility of storing different attributes of adjacent polygons by extending the quad record (TQuad) of the quad-edge structure with an additional attribute field, which adds four attributes to each edge of the mesh (two for the Delaunay edge and two for its dual edge).

However, the most important difference between these two diagrams is that the LSVD separates objects from topology, while the CDT "mixes" them. Objects in the LSVD are well defined distinct entities with proximal regions associated with them and relations between them are defined by the edges of the triangulation. In the CDT the information about objects and topology is all mixed together "inside" edges of the triangulation.

Additionally, the line segment Voronoi model can be used for curves, circles and arbitrarily objects as well (Alt and Schwarzkopf (1995), Gavrilova and Rokne (1999)), not only for straight lines, which is not possible for the CDT based on straight triangulation edges.

Concluding, it appears that the line segment VD is more suitable for spatial analysis than the CDT (as noted in Gold and Dakowicz (2007)), because apart from defining relationships between

objects, it provides proximal regions for separate elements of the map and for compound objects, such as polygons representing buildings. It also a more elegant solution, as it separates objects from topology. Thus the LSVD based models will be used in this chapter to illustrate various spatial operations.

6.1.2 Discrete Objects

In 2D we can distinguish four main types of discrete objects: points, lines, polygons and mobile objects. Points can be converted to fields by calculating their Voronoi Diagram. Figure 6.2 presents a map constructed from imaginary locations of mail boxes. Every location of the map is assigned to the nearest mailbox and the neighbourhood relationships of mailboxes can be easily defined.

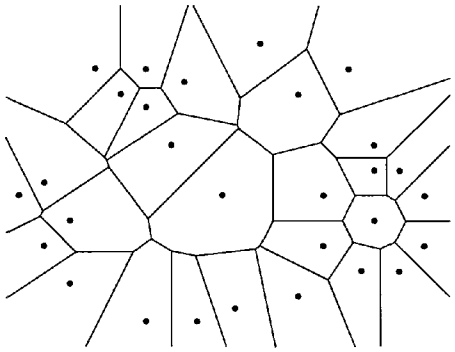


Figure 6.2: The Voronoi Diagram created for the points representing locations of mailboxes.

Lines can be converted to fields by constructing the CDT or LSVD with segments representing input lines, which establishes relationships between them, and in the case of the LSVD, associates proximal regions with each segment.

Polygons can be converted to fields, creating the CDT or the LSVD from nodes and segments defining their boundaries, as explained before in Section 6.1.1.

Finally, there are two forms of interaction of mobile objects with the mesh (Gold (1994)). In the "plane model" objects are not added physically to the mesh, but "fly above", retrieving information from the mesh when necessary using queries. Another is the "boat model", in which mobile objects are embedded into the Voronoi diagram. They interact with other mobile objects and components of the mesh (buildings, shoreline) in the mesh and their locations are changed using the moving point method. Figure 6.3a shows a LSVD map created for building objects. There are also four point objects, which could be cars or people incorporated into the mesh. Objects c and d share a common Voronoi edge. Figure 6.3b shows the same map after the point c was relocated to a different position using the method MovePoint, so it is not a neighbour of d anymore.

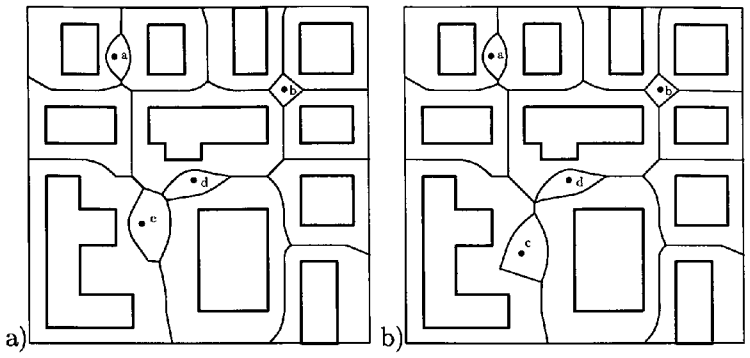


Figure 6.3: Moving point objects between polygons (boat model). a) The initial situation with four mobile points a, b, c and d. b) The situation after relocating the point c.

6.1.3 Networks

Networks can be modelled in several ways using the VD/DT, the CDT or the LSVD. Figure 6.4a shows a drawing of an imaginary road network. Such an image is usually converted to a digital network by digitizing it. However, this often leads to errors at junctions (overshoots and undershoots), where several segments of the network meet, and the challenge is to assure that they are joined. A common solution is to use a snapping technique so two nodes at a specified distance are merged into one.

Using the MovePoint based kinetic algorithms to constructs CDT or LSVD network models eliminates problems at the junctions, as all objects have buffer zones associated with them and are automatically merged when disks of moving points collide with their buffer zones. Additionally each segment of the network can have an attribute assigned, such as traversal cost or road width. When using the LSVD, different attributes can be assigned to adjacent half-lines of each line segment, so traversing an edge in the opposite direction can have a different cost. An edge can be also made directed, setting for example the attribute of one of its half-edges to "zero". Similarly, in the CDT attributes can be assigned to each quad of the quad-edge.

Figure 6.4b shows the LSVD created from the road network data. The model was created using the MovePoint method, so each part of the network is represented by a line segment object and they are correctly joined at the junctions. The topology is defined and various analyses can be performed. Various proximity relationships are readily available, so it is trivial to calculate a distance from the point x in Figure 6.4b to adjacent road segments or to locate the nearest road.

Figure 6.4: A road network. a) The outline of the network. b) The LSVD of the network.

Another way of constructing networks is to extract dense samples of the model (Figure 6.5a) and create a simple VD and DT (Figure 6.5b). If the sampling of the network is dense enough then the crust extracted from the DT refers to the elements of the network and the skeleton approximates the medial axis of "open polygons" between crust branches. When modelling river data this skeleton approximates the configuration of the watershed (Figure 6.5c and thicker lines in 6.5b). Such a network can be traversed from any point using the topology of the diagram, so various analysis methods are possible (described later in Section 6.3).

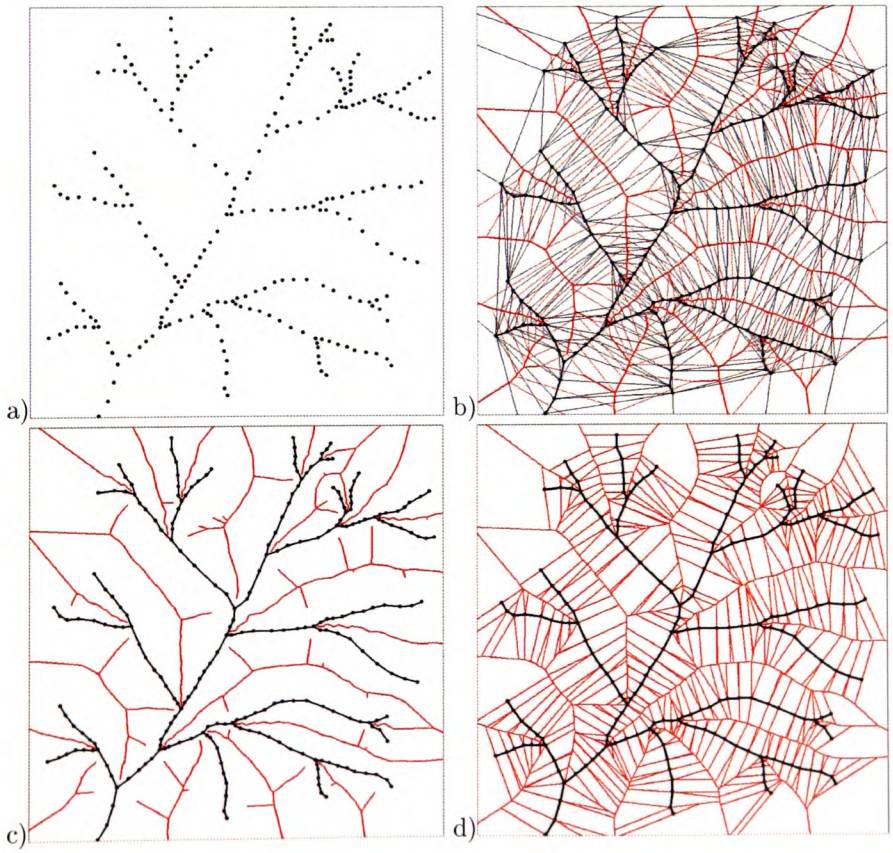


Figure 6.5: A river network. a) The samples of the network. b) The VD and DT with crust and skeleton marked with thicker lines. c) The crust and skeleton extracted from the VD/DT. d) The LSVD created from the samples.

6.1.4 Polygonal Maps

Space exhausting polygonal maps are discrete field models, with all locations inside one polygon having the same attribute value. Figure 6.6a shows a sample polygon map depicting postcode boundaries. Figure 6.6b shows the LSVD created from samples of boundaries of the polygons. Each polygon's boundary consists of a chain of connected line segments. Since two adjacent polygons share a common boundary the "two half-lines" representation of line segments in the LSVD (described in Section 5.2) becomes very valuable as one half-line refers to one polygon while its opposite half-line refers to the adjacent polygon. Each half-line can have different attributes, such as a postcode, the owner's name or the area of its Voronoi cell. Such a representation of polygons, together with the quad-edge data structure of the mesh and topology relationships between objects available via Delaunay edges of the LSVD, turns traversal along the boundary of each polygon and from one polygon to another, into simple operations on quad-edges.

Figure 6.6: A postcode sector boundaries map. a) A polygon map. b) The LSVD created from the data.

A frequent problem of polygonal map construction and merging is the possible generation of sliver (or 'weird') polygons (Chrisman (2002); Heywood et al. (2002)). When the same boundary is digitized separately for each polygon or when the data come from different sources, it often happens that instead of a single line where two polygons meet exactly, two different lines are created. These lines intersect each other and there are additional sliver polygons between them, as shown in Figure 6.7.

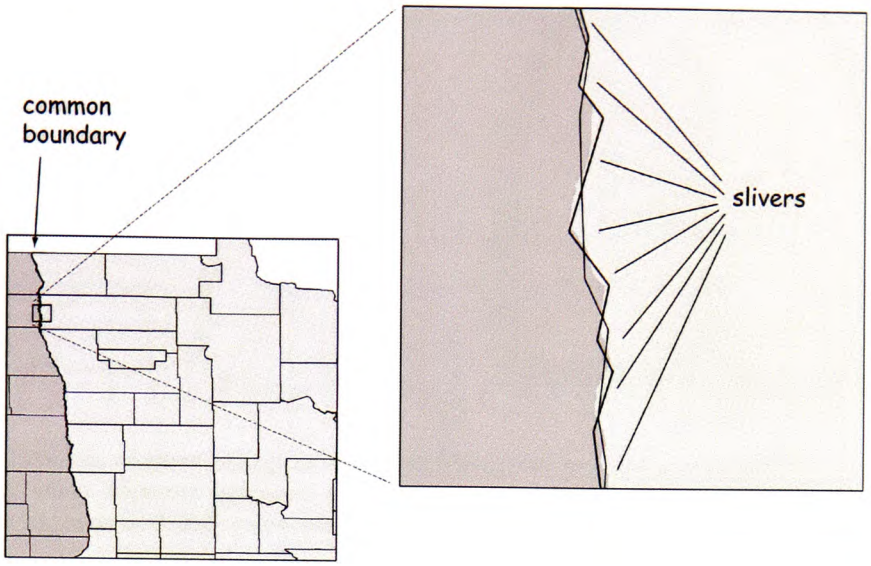


Figure 6.7: Sliver polygons when combining two maps (after Bolstad (2003)).

A common solution to the sliver polygons problem is using a tolerance in the generation or overlaying process (Longley et al. (2001); Chrisman (2002)). If two lines fall within a specified distance they are merged into one line. As the result one of the lines is moved and if the tolerance value is relatively large the shape of the line may change substantially.

When constructing or merging two CDT or LSVD layers the sliver polygons problem is handled automatically by the MovePoint procedure, as a specified disk radius is associated with each point, constrained edge or line segment and line segments are "snapped" if they overlap. This also solves the problems of overshoots and undershoots in the map generation process (when junctions of polygons are not precisely defined), but are joined properly by snapping them in the MovePoint operation.

Figure 6.8a shows two sample polygons that are adjacent, but due to the data coming from two sources their common boundary overlaps, with many sliver polygons present between them. Setting a relatively large disk size before the map construction prevents introducing sliver polygons (Figure 6.8b). After creating the first polygon, the insertion of samples of the second polygon along the common boundary is attempted, but all of them are snapped to the existing boundary of the first polygon. Additionally, some of the boundary points of the original polygon are not inserted, as the length of some edges was smaller than the doubled disk size.

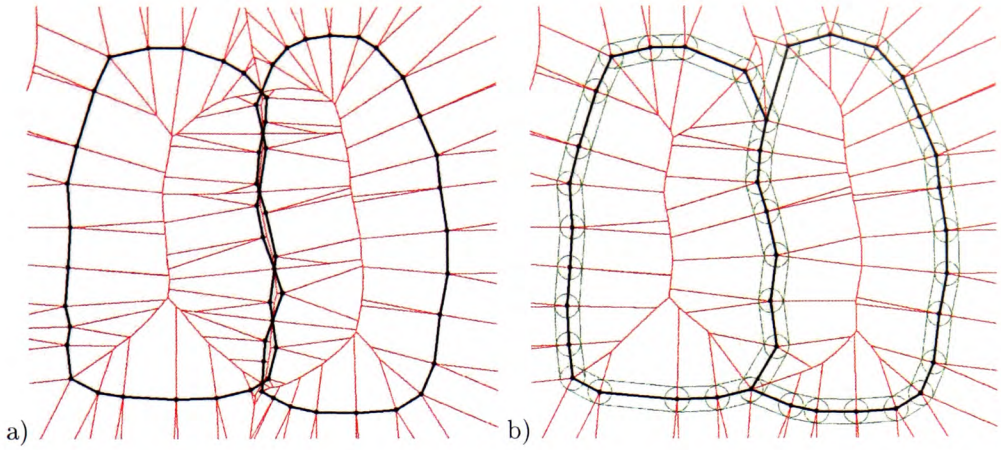


Figure 6.8: Problem of sliver polygons when combining two polygons using LSVD. a) Sliver polygons present when adjacent polygons are created with a small disk radius value. b) Polygons created with an increased disk value without sliver polygons.

6.1.5 Surfaces

Surfaces is the fourth category of spatial models discussed in this thesis. These are field models having attribute values available at all locations. Traditionally they are represented as sample data points, contours or grids. The sample data points (Figure 6.9a), which are discrete objects, can be converted to a field surface model by generating the VD or the DT from them (Figure 6.9b). This adds topology to the model and defines neighbourhood and proximity relations.

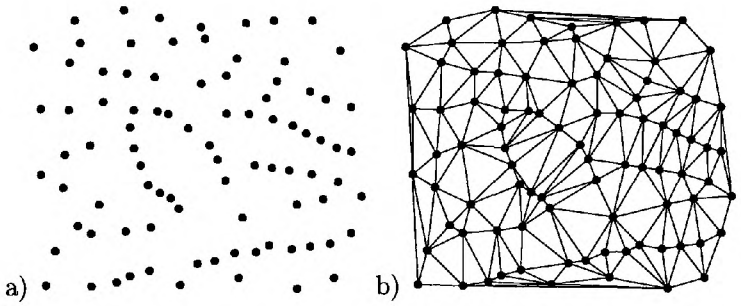


Figure 6.9: Elevation data. a) The sample points of the terrain. b) The TIN model.

Contours are usually represented as raster images with drawn lines or as vectors with each contour being a distinct object consisting of series of connected nodes (such as ESRI shapefiles). Raster contours can be converted to fields (as described in 6.5.2) by extracting their samples (Figure 6.10a), triangulating them into a Delaunay TIN model (Figure 6.10b) and assigning elevation values to nodes. Samples of contours stored in vector format can be used directly to create TINs.

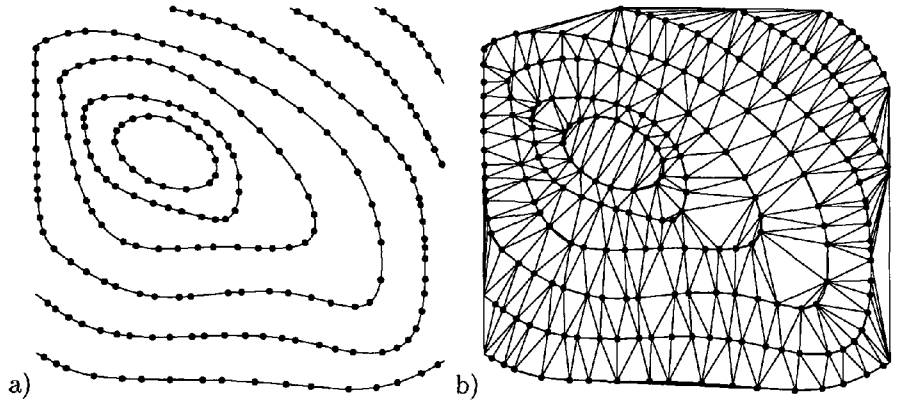


Figure 6.10: Contour surface. a) The contour lines and their samples. b) A TIN created from samples of contours.

Grids (Figure 6.11), which are raster fields can be converted to vector fields making the VD of the centres of grid cells (Figure 6.11b) or the TIN (Figure 6.11c). Alternatively, the locations of intersections of cells boundaries can be used, instead of the centres of the cells.

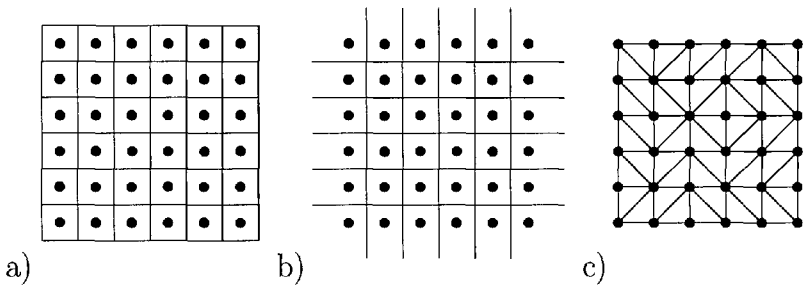


Figure 6.11: Grid surface. a) The raster grid with marked centres of cells. b) The VD created from the centres of cells. c) The TIN created from the centres of cells.

The VD/DT representation of surfaces and previously described discrete point objects is technically the same. The only difference is the way they are interpreted. The surface is defined by the data structure (the VD/DT) and the interpolation algorithm for finding values for points between the given points, while the VD/DT of discrete point objects is just a collection of connected nodes with attributes, without any additional information.

6.2 Applications for Discrete Objects

As was mentioned before, there are four main types of discrete object maps: points, lines, polygons and mobile points. They can be converted to fields making a simple Voronoi diagram (for points) and a CDT or LSVD (for lines and polygons) so each line or polygon can be represented using line segments or constrained edges. Mobile objects can be inserted into diagrams and managed using the kinetic MovePoint mechanism.

Applications for Voronoi diagrams created for discrete objects include geological mapping using skeleton operations (Section 6.2.1) and feature extraction from scanned maps (Section 6.2.3). The LSVD maps created for polygonal objects can be used to define/detect neighbourhood relationships of buildings or to extract navigation paths between them (Section 6.2.4). Buffer zones (Section

6.2.2) can be obtained for objects (as well as for networks and polygonal maps) and used as new maps for further analysis. Examples of applications utilizing mobile objects include the Marine GIS for ship movement simulation (Section 6.2.5) and collision avoidance and Free-Lagrange flow for simulating global tides (Section 6.2.6).

6.2.1 Geological Mapping

Figure 6.12 shows the VD of a set of rock outcrops of various types distinguished by the numbers assigned to different locations. The heavy boundary lines separate cells with different values of labels, and thus give an approximate geological map. Such a boundary is termed a "labelled point skeleton" (mentioned in Section 3.8) and it consists of Voronoi edges that are the duals of triangulation edges connecting nodes with different labels (Gold et al. (1996)).

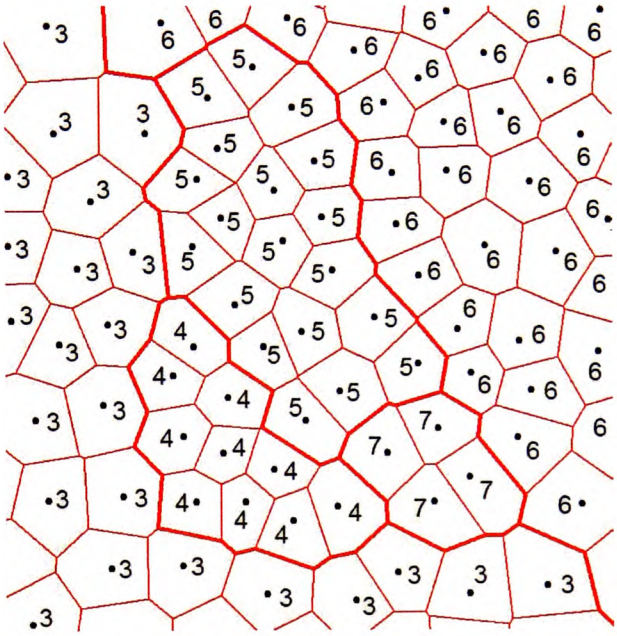


Figure 6.12: A labelled point skeleton separating different types of rocks.

6.2.2 Buffer Zones

Buffering creates a zone of interest of a specified size around an entity (Heywood et al. (2002)). This operation is used for example to find hotels within a specified distance from the main road (by creating a buffer zone region around the road and then locating hotels using a "point-in-polygon" operation) or determining the areas adjacent to the river network. Buffer zones can be calculated for discrete objects, networks and polygons.

If the entity is a point, then buffering creates a circular zone around it, as in Figure 6.13a. If buffer zone circles overlap then the overlap areas are removed, as in Figure 6.13b. For lines and polygons buffering creates new areas, being constant width corridors with circular arcs around endpoints and corners (Figure 6.13c and d).

The operation is possible for vector and raster representations (Longley et al. (2001)). In the raster representation the cells are classified according to whether they are inside or outside the buffer. In the vector case the result is a new polygon or polygons, enclosing the original

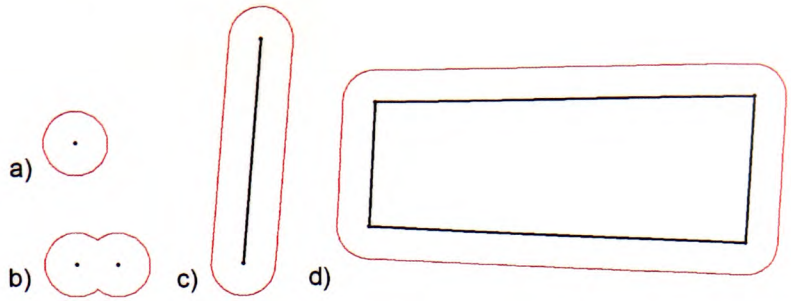


Figure 6.13: Buffer zones. a) A single node. b) Two nodes. c) A line segment. d) A polygon.

data. Buffer zones in the VD can be calculated by examining each Voronoi cell in turn. If it was generated by a point object a circular arc of the required radius is drawn within the cell around the generator. If the generator is a line segment a parallel line is drawn within the cell at the required distance from the generating segment. Buffers can be used as a temporary aid to other queries or as separate objects added to the database.

Although this section concerns discrete objects, an example of a buffering operation is given on a network map, as using the LSVD this operation is performed in the same way for discrete objects, networks and polygons. Figure 6.14 shows a process of extracting buffer zones from a network map represented by the LSVD (Figure 6.14a). Figure 6.14b shows buffer zones drawn on top of the line segment Voronoi diagram. Figure 6.14c shows a new LSVD map created from nodes defining the buffer zones of the road network. The circular arc in the bottom left corner is broken into line segments approximating its shape.

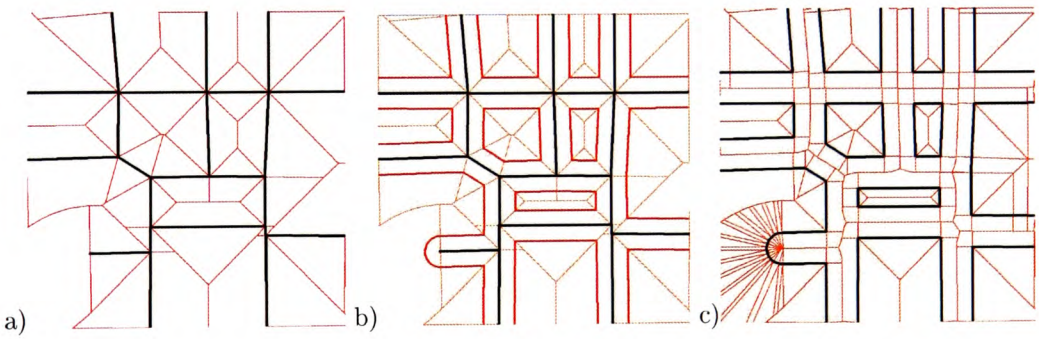


Figure 6.14: Buffer zones of a road system. a) The LSVD. b) Buffer zones of network segments drawn on top of the LSVD. c) A new LSVD map created from the buffer zones

6.2.3 Rapid Digitizing, Scanned Maps and Text Recognition

An effective method to construct polygon maps rapidly, e.g. for annual forest mapping is to generate points with labels by "rolling" the digitizing cursor round the interior of each polygon, as in Figure 6.15. Such points can be used to construct a simple topologically-connected polygon map by generating the VD and extracting the labelled skeleton (Voronoi edges separating different labels) (Gold et al. (1996)). Such an approach reduces the number of errors (such as overshoots or undershoots) often introduced by traditional map digitizing.

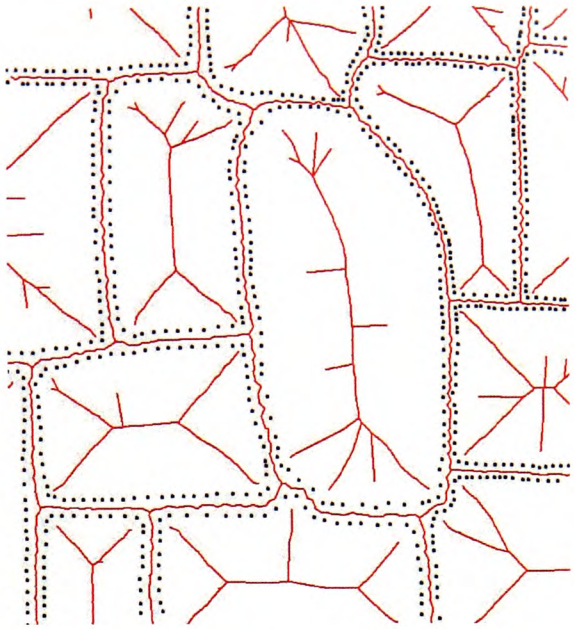


Figure 6.15: Rapid digitizing of a polygon map.

A similar approach can be used to extract features from scanned maps, as described in Gold and Snoeyink (2001). A Voronoi diagram is created by inserting points densely along all "black/white" boundaries. This generates skeletons for white (polygons) and black (linework) portions of the map and the crust approximates the centre line of map features, as shown in Figure 6.16. Scanned text can be processed using the same edge-detecting technique.

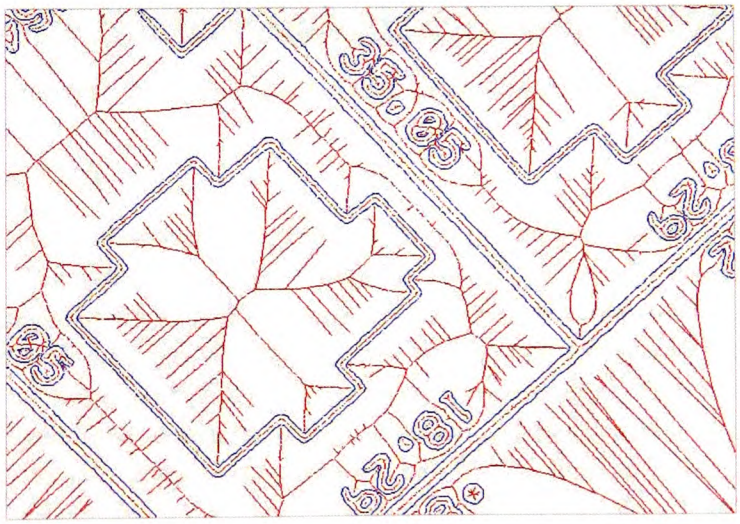


Figure 6.16: A scanned cadastral map with the crust and the skeleton drawn (from Gold (1999)).

6.2.4 Discrete Polygons as Proximity Maps

A Line Segment VD created from discrete polygons (shown in Figure 6.17) provides much more useful information than a set of unconnected polygons itself. Each building is defined by a set of connected line segments. Each line segment object consists of two half-lines (Section 5.2), each being a generator of a Voronoi region. Thus for each line segment there is one Voronoi region inside

the polygon and one outside. Their interiors can be easily extracted, as well as exterior cells which give a good definition of proximity to other buildings. The connectivity of buildings is defined by triangulation edges connecting line segments from different buildings, whose dual Voronoi edges are shared by Voronoi regions of two different polygons. The borders between exterior cells of buildings (the central skeleton) define paths between them and can be used for navigation, route planning or calculating widths of passage.

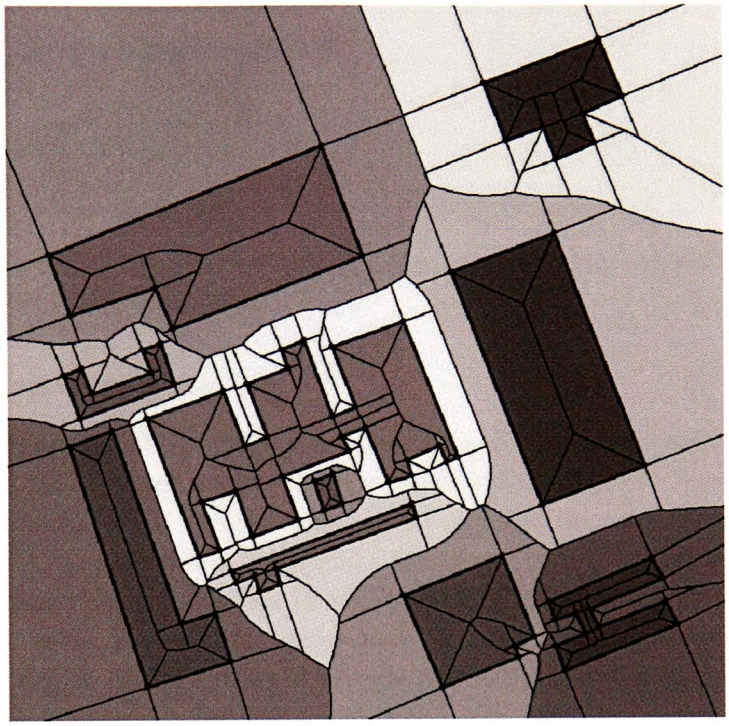


Figure 6.17: Neighbour relationships between buildings in a LSVD.

6.2.5 Marine GIS

Recently kinetic Voronoi diagrams were used as a basis of a new type of GIS system for maritime navigation safety (Goralski et al. (2007); Goralski and Gold (2007)). The spatial relationships of ships and other navigational objects (buoys, lighthouses, restricted areas or wrecks) are maintained using a quad-edge structure based Voronoi diagram. The ships are represented in the VD as moving points and their movement maintained with the MovePoint procedure. The shore line is a part of the diagram and it can be represented as strings of nodes depicting the outline of the land (a kinetic VD as in Figure 6.18) or by line segments (a kinetic LSVD). The ships (two of them in Figure 6.18) are the objects moving within the sea areas bounded by the constant shore lines (or specified "sea roads" - fairways) and their positions are updated using standard onboard AIS transponders and the moving point Voronoi mechanisms. The spatial relationships between all objects are used to detect and avoid collisions of ships. Such a Marine GIS armed additionally with 3D visualization provides navigational and decision support to navigators to help tackling the main cause of marine accidents - human error.

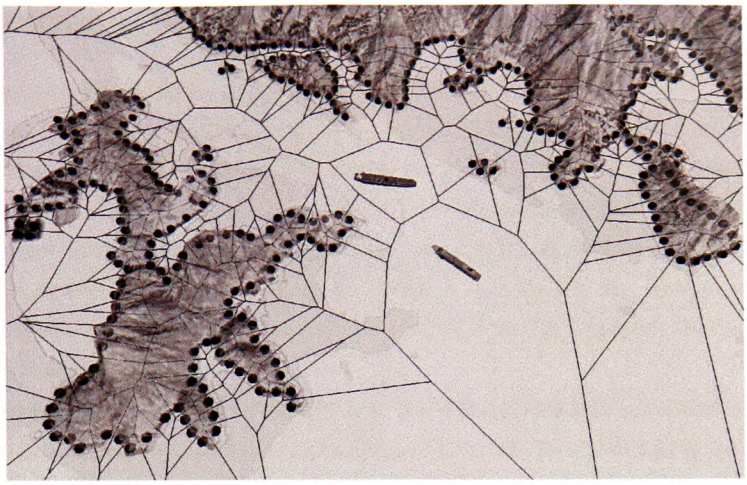


Figure 6.18: Marine GIS showing the underlying Voronoi diagram of the shore line and two moving ships (moving points rendered as ships).

6.2.6 Free-Lagrange Flow

The Free-Lagrange method (FML, Fritts et al. (1985)) is one of the fundamental approaches for simulation of fluid flow. It uses a set of discrete points with a mass and velocity moving freely and interacting. Previous FML implementations were very slow because adjacency relations had to be recalculated at each iteration of the process. Mostafavi and Gold (2004) discussed modelling FML using the kinetic Voronoi diagram in which all topological changes are managed locally. The outlines of the continents were marked by double lines of fixed points, so that the inner line would not be changed. A regular pattern of Voronoi cells was placed throughout the oceans, each representing a fixed volume of water (see Figure 6.19). The cells were moved using the moving point mechanism, greatly improving the efficiency of the simulation.

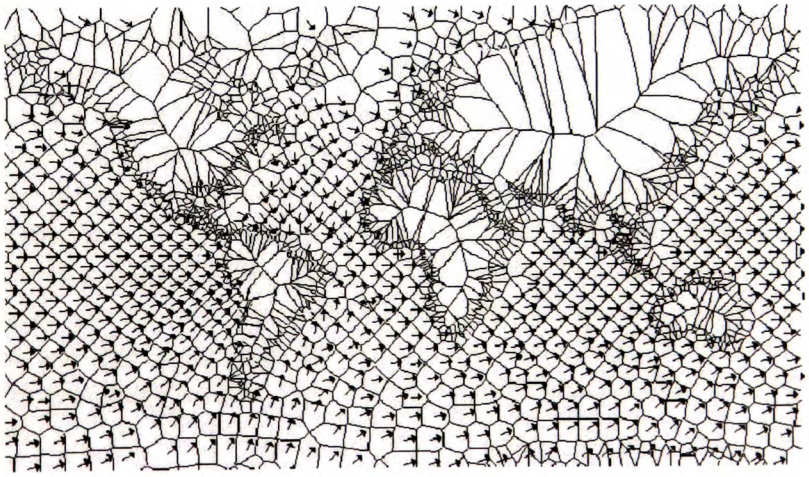


Figure 6.19: Free Lagrange flow (from Mostafavi and Gold (2004)).

6.3 Applications for Networks

Networks surround us everywhere. Roads, power-lines and rivers are the most common examples. They can be modelled using segments (called edges or links) and locations of their joins can be marked with points, often called nodes. In order to perform analysis on networks (navigation) the connectivity of segments has to be assured. Edges of networks can be directed (one-way) or undirected (two-way) and there can are attributes associated with them and with nodes (name, length, travel time, travel cost) (de Smith et al. (2008)). Attributes can be also assigned to nodes of the network. Additionally some nodes can be specified as centres and the flow between them can be modelled (Bolstad (2003)).

Networks represented by Voronoi diagrams can be used to tackle classic network problems such as network routing, tracing and allocation (Section 6.3.1), and standard graph theory algorithms such as Breadth First Search, Depth First Search or Minimum Spanning Tree (Sedgewick (1992)). Each element of the network can be represented as a constrained edge or a line segment object having a weight value assigned (attribute). Directed edges in the LSVD can be easily handled since line segments consist of two adjacent half-lines. Two-way parts of the network can be handled using different weight values for opposite half-lines, and one-way segments can be handled marking one of the half-lines with a "zero" weight for example. Similarly in the CDT the quad elements of the quad-edge can have different weights assigned and used to model directed networks. When representing river networks the VD can be used to generate watersheds (Section 6.3.2) and build a cumulative catchment model (Section 6.3.3). Additionally in all Voronoi based network models various proximity relationships are readily available within the 2D space.

6.3.1 Network Analysis

There are three main types of network analyses: network tracing, network routing and network allocation (Korte (2001)).

Network tracing is an operation used to determine a path through the network using criteria provided by the user in order to trace the flow of people, goods or information (Heywood et al. (2002)). It deals with queries such as "Is it possible to reach node A from node B?" or "What nodes can be reached from a given node?" It can be used to find telephone customers affected by a broken cable or to determine streams contributing to a reservoir.

Network routing determines the optimal path based on numerous criteria, such as the shortest distance, the fastest route or minimum cost. The analysis can check for route restrictions, which can be one-way streets, no turns or blockages. Network routing examples include reaching accidents sites by emergency services, car navigation to avoid traffic or use the shortest way and planning routes of express deliveries. Figure 6.20 illustrates the concept of transit costs. It shows an example network with traversal costs assigned to each of its segments and the least cost path between two nodes is marked.

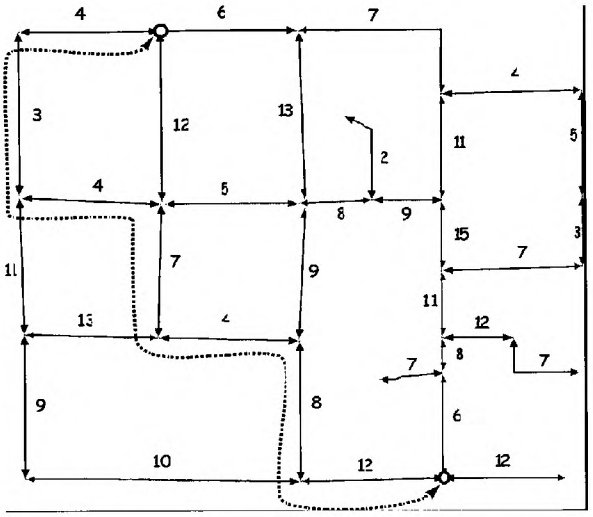


Figure 6.20: The least cost path in the network (from Bolstad (2003)).

Figure 6.21 shows the network from Figure 6.20 represented by a LSVD. Each segment of the diagram has a traversal cost assigned as its attribute. Two nodes are marked respectively as the origin and destination of the traversal and the least cost path can be found between them using the LSVD topology and shortest path finding algorithms.

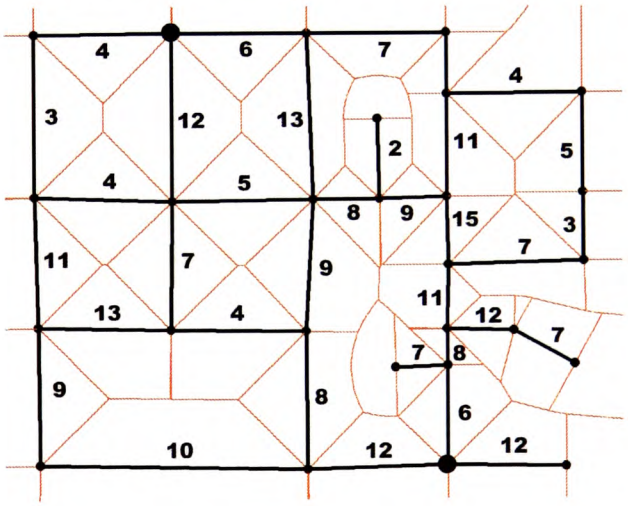


Figure 6.21: The network from Figure 6.20 represented by a LSVD.

Network allocation methods assign portions of the network to "the centres of supply" or "the points of destination" defined in the network (Bolstad (2003)). Routes are calculated outwards from each centre and network elements are assigned to "nearest" centres, according to the distance, time or cost. A common example is assigning children to schools (supply centres). If we assume no limit to the number of students each school can accept, then each part of the network is assigned to the nearest school. More complex analysis can also take into account the capacity of schools and the travelling time instead of the distance. Another common example is marking streets withing a five minute walk of a specified bus stop (destination point). Such a situation is shown in Figure 6.22 where four bus stops are marked as allocation centres and network elements are assigned to "nearest" centres.

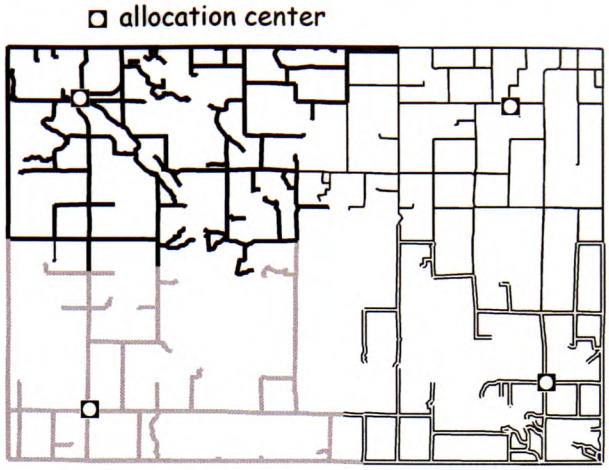


Figure 6.22: Assigning network elements to allocation centers (from Bolstad (2003)).

6.3.2 Watershed Generation

A watershed is an area that contributes flow to a point on the surface (Bolstad (2003)). In other words the area above a specified location that drains to that location is its watershed. They can be very small near lines ridges or summits of hills, be of zero area on local high points, or large in the valleys or in river channels. The river network data, even without any elevation data, provides enough information to construct a very reasonable watershed model (Gold and Dakowicz (2005)). The idea of Blum (1967), who assigned the elevation values for the medial axis basing on the distance from the curve only, comes in handy here. The watershed is equidistant to the river, so can be approximated with the medial axis. Slopes on each side of the watershed are often similar.

Figure 6.23 shows connected samples of an imaginary river basin. The data is enclosed by a frame, which simplifies the generation process, while in the real world branches of rivers "interfinger". All data points have the same elevation value in this example, although variable values can be handled as well.

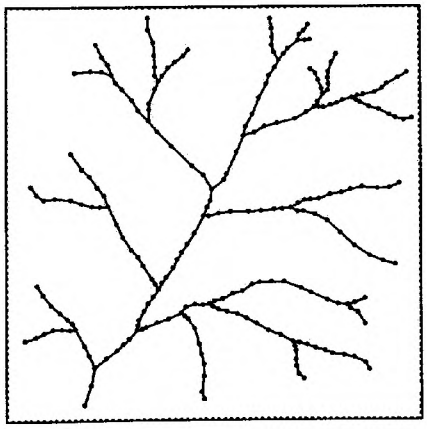


Figure 6.23: Samples of a river network.

Figure 6.24a shows the TIN model constructed from the samples (black lines). The Voronoi diagram is computed as well (brighter lines) and the skeleton, being a subset of the VD, is extracted. Since the river is densely sampled the crust (thick black lines) reconstructs the shape of the river.

At the same time the skeleton approximates the ridges of the watershed. Figure 6.24b shows both the crust and the skeleton extracted from the Delaunay triangulation and the Voronoi diagram respectively.

The skeleton points can be used to construct the ridges of the watersheds. Skeleton nodes are vertices of the Voronoi diagram. The elevation value of each skeleton node can be assigned based on the radius of the circle having that node as its circumcentre, so skeleton nodes being circumcentres of small triangles have a small elevation value, and those associated with large triangles have the largest elevations.

The computed elevation values can be scaled to adjust them to the model coordinates. Also if the river network nodes have variable elevation values the resulting watershed elevations have to be adjusted accordingly, considering the initial elevation of the skeleton points (which can be initially interpolated using the river elevation values).

Figure 6.24b shows a 3D view of the resulting model with the skeleton draped on it. The visual inspection of the model tells us that it looks reasonable and matches our perception of the intended watershed.

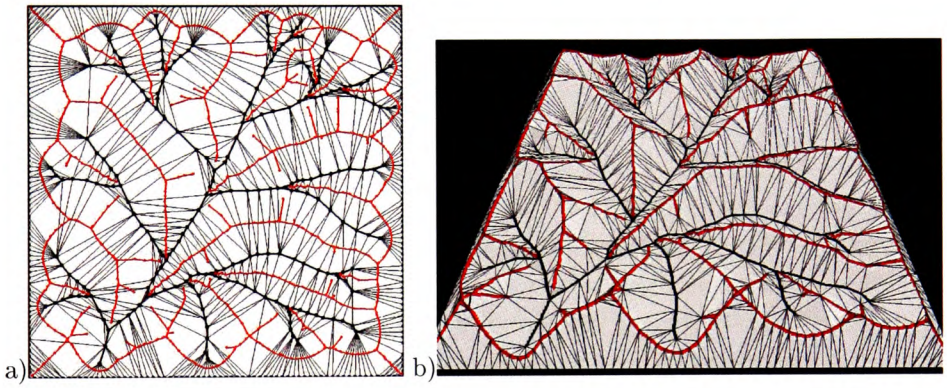


Figure 6.24: The triangulation of the river data. a) The TIN model with the Voronoi diagram and the crust and skeleton marked with thicker lines. b) The 3D view of the enriched river data with generated watersheds. (from Gold and Dakowicz (2005))

6.3.3 Cumulative Catchment Areas

Assuming a fixed rainfall throughout the map, the river network data can be used to compute cumulative catchment areas (Gold and Dakowicz (2005)). In the TIN model constructed from the samples each node of the river network has a Voronoi cell associated with it. When the sampling is dense enough the edges connecting samples form the crust and the dual Voronoi edges of the remaining Delaunay edges form the skeleton that can be later used to generate the watersheds. The volume of water in each cell is proportional to its size. The cumulative sum of these volumes downstream gives an estimate of the total water flow at any river node. This can be seen in Figure 6.25, where the height of the bars represents the volume of water. This cumulative catchment model provides a first-order approximation of flow, based on the drainage network alone.

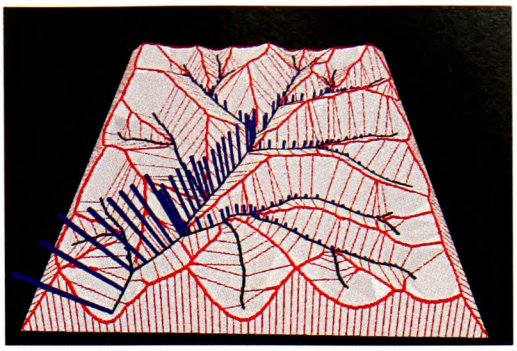


Figure 6.25: The cumulative catchment areas. (from Gold and Dakowicz (2005))

6.4 Applications for Polygon Maps

Space partitioning polygon maps, which are discrete fields, are mainly used as layers in "query" and "merge" operations described later in Section 6.6. Applications of single layers include traditional GIS operations such as reclassification (Section 6.4.1) and dissolving (Section 6.4.2).

6.4.1 Reclassification

Reclassification is one of the most common operations on a single map layer. It can be considered of as "the purposeful recolouring of maps" and it "involves the simple repackaging of information on a single overlay and results in no new boundary delineations" (Berry (1993)). Basically, it is an operation on attributes of the map and is usually used to group objects with specified properties for display or map production (Bolstad (2003)). Selected object can be displayed with the same colour so they can be identified as a group. It is also used to change attributes, often using classification tables specifying the classification assignment for each class of the map. An example of such an operation is shown in Figure 6.26 with the input and the resulting map processed using the classification table defining input and output classes.

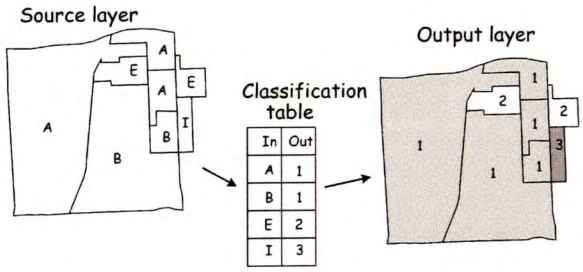


Figure 6.26: Reclassification of a map using a classification table (from Bolstad (2003)).

The simplest classification operation is perhaps the binary reclassification, which assigns objects of the map to two classes, depending on the specified criteria. One common example is a map of the results of the US presidential elections, grouping states voting for each of the candidates. This is shown in Figure 6.27 with states coloured in one of the two colours, depending on the winning candidate.

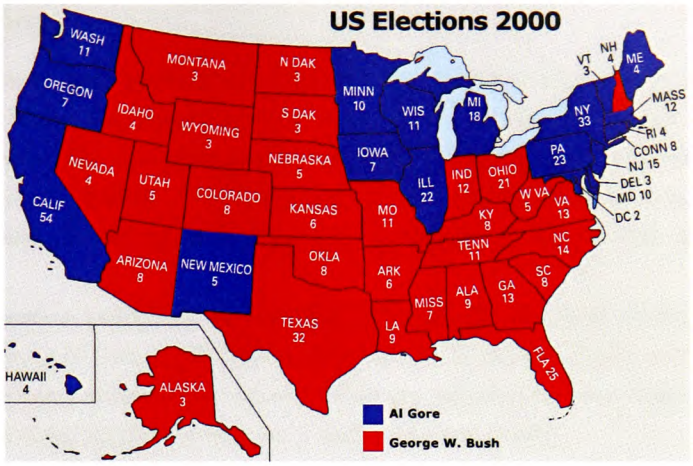


Figure 6.27: Reclassification in a map of the USA presidential elections (from Wiki Commons).

Reclassification operations can be performed on any polygonal maps represented by the LSVD. Each region is bounded by a closed chain of line segments and all Voronoi cells inside each polygon (associated with half-lines of line segments) share the same attribute value (which can be the population density). Figure 6.28a shows a LSVD polygonal map of imaginary electoral wards boundaries, with the attribute depicted by the intensity of gray colour. Figure 6.28b show the same map after white regions are reclassified to the same group with the darkest one, which results in changing colours of four regions (no changes in the shape of the polygons).

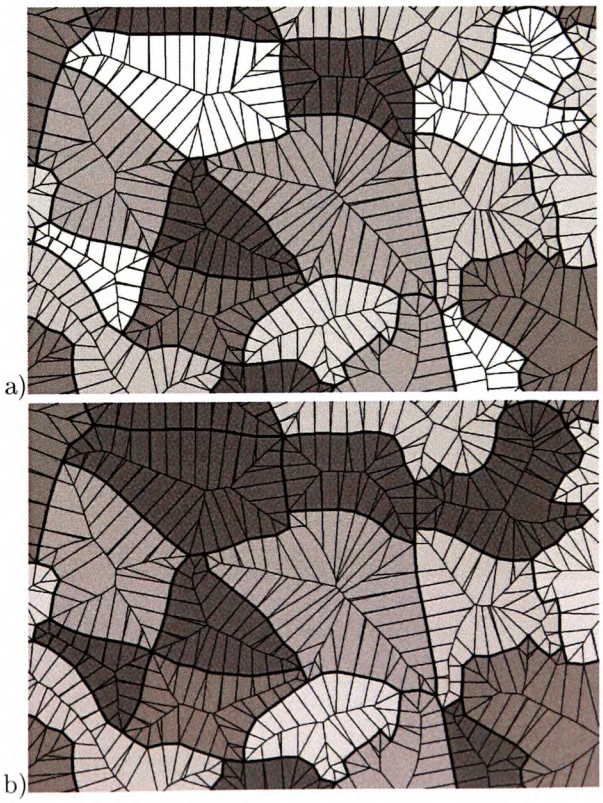


Figure 6.28: Reclassification of a LSVD polygonal map. a) The initial map. b) After the reclassi fication.

6.4.2 Dissolving

Another popular single layer operation is dissolving illustrated in Figure 6.29. Its main purpose is combining features in the map according to specified rules. These rules usually concern equality or similarity of attributes. The operation leads to merging adjacent features by dissolving boundaries between them (Bolstad (2003)). The operation can be performed by testing all boundary lines between polygons and dissolving these separating polygons with attributes satisfying the query rule.

Figure 6.29a shows a polygonal LSVD map of imaginary electoral wards having population density as the attribute of each region represented by the intensity of gray colour. Regions *a*, *b*, *c*, *d* and *e* sharing the same value of the density value are adjacent to each other and can be merged into one region, the same applies to regions *f* and *g* (Figure 6.29b).

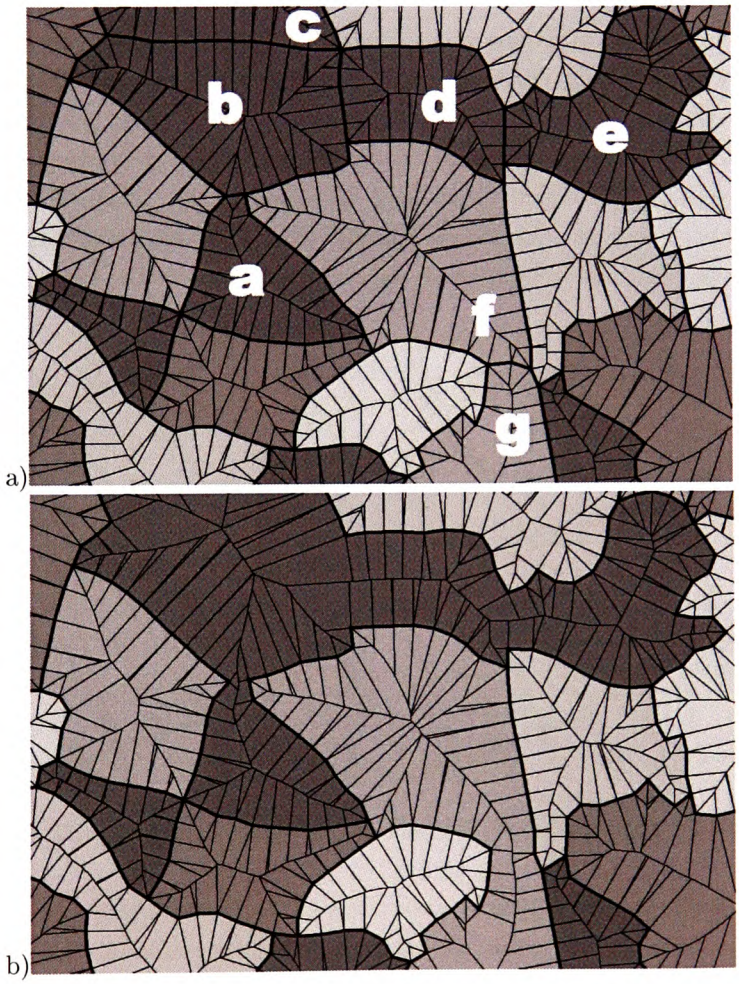


Figure 6.29: Dissolving in a LSVD map. a) The initial map. b) After merging two groups of regions.

6.5 Applications for Surfaces

Surfaces are based on points with the elevation value as their attribute. These points come from different sources: they can be elevation point directly surveyed in the field, obtained by scanning

the Earth using the LIDAR (Light Detection and Ranging) technique or extracted from contour maps. They are converted to fields by creating a VD/DT. When samples come from contours, additional processing of the DT is necessary to get rid of "flat triangles" connecting samples with the same elevation (Section 6.5.2). Additionally, scanned and digitized contours can be smoothed using the skeleton method, described in Section 6.5.1. Various interpolation methods can be used to convert these diagrams into continuous surfaces (Section 6.5.3) by estimating elevation values at any location. The DT based terrain model can be a base for a surface modification system, where the relief is changed locally using a cutting knife (Section 6.5.4). The VD of the surface can also be a base for an efficient method of runoff modelling, where each Voronoi cell is a bucket storing water and water is moved between buckets (6.5.5).

6.5.1 Contour Smoothing

Scanned and digitized contour lines are often used as a source of the data for digital terrain models. After triangulating densely sampled contours there are often many irregularities in the shape the computed crust (edges connecting dense samples). The skeletons extracted from Voronoi diagrams are very sensitive to perturbations of the boundary points of the crust and can be used for smoothing the crust. Misalignment of crust nodes results in branching of the skeleton. Such branches frequently consist of a single Voronoi vertex only, as in Figure 6.30a, where in a Delaunay/Voronoi pair of edges ac and uv, the edge uv belongs to the skeleton.

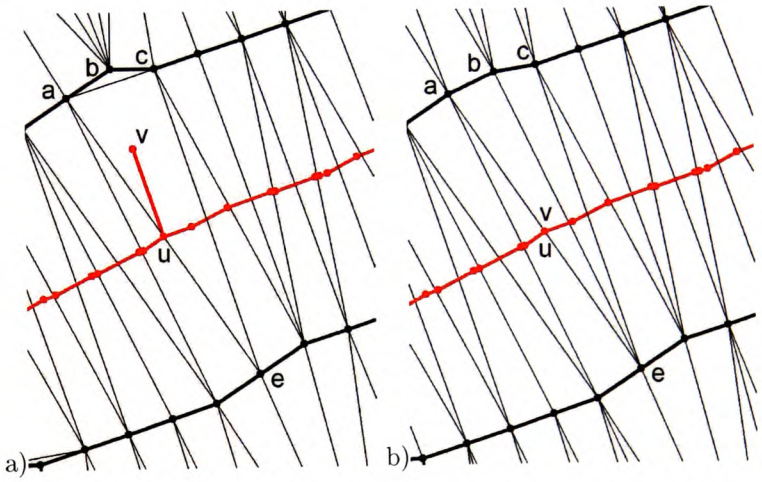


Figure 6.30: The crust generalization. a) The initial configuration. b) The result.

A relatively simple adjustment of positions of crust points eliminates such single-point skeleton branches. Figure 6.30b shows the result of relocating the node b of the crust onto the circle ace centred at the point u with the radius au, as described in Thibault and Gold (2000). As a result four nodes a, b, c and d become cocircular and the edge ac is switched to be. The length of the dual Voronoi edges of be is zero so u and v share the same location. This operation makes the skeleton simpler and the crust smoother.

Figure 6.31a shows the crust and skeleton of a fragment of a map created from densely digitized contours. There are many unwanted small skeleton branches created due to small perturbations of digitized samples. Figure 6.31b shows the same map after smoothing the crust by removing all single-point branches of the skeleton.

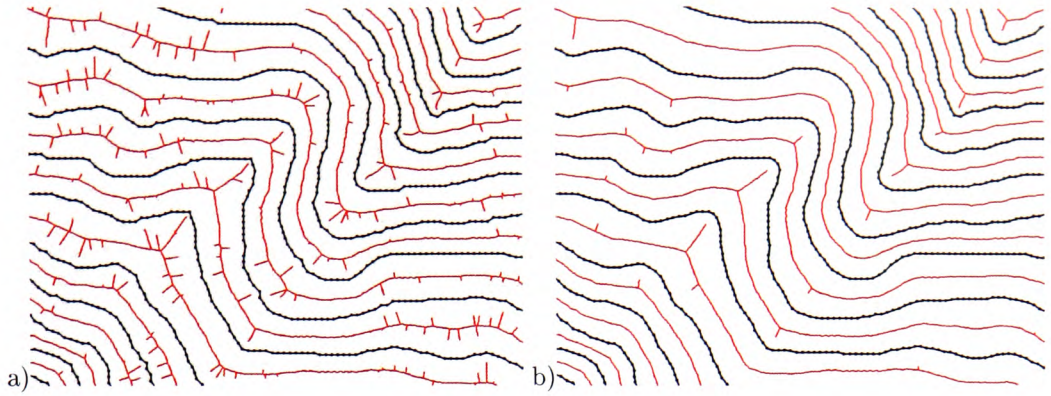


Figure 6.31: Generalization of contours. a) The initial map. b) The result.

6.5.2 Terrain Modelling[1]

The quality of terrain models is becoming more and more important as applications demanding better surface orientation information than is available from traditional interpolation techniques are developed, such as runoff modelling. Recent work on the automatic reconstruction of curves from point samples, and the generation of medial axis transforms (skeletons) has greatly helped in expressing the spatial relationships between topographic sets of contours. With these techniques the insertion of skeleton points (Amenta et al. (1997), Gold and Snoeyink (2001)) into a TIN model generated from samples of contours guarantees the elimination of all "flat triangles" where all three vertices have the same elevation. Additional assumptions about the local uniformity of slopes give us enough information to assign elevation values to these skeleton points (Dakowicz and Gold (2002b)).

The input data is in the form of contour lines that are sufficiently well sampled – perhaps derived from scanned maps. Despite modern satellite imaging, much of the world's data is still in this form. They are subjective, the result of human judgment at the time they were drawn. Thus they are clearly intended to convey information about the perceived form of the surface at a particular scale – and it would be desirable to preserve this, as derived ridges and valleys. Figure 6.32a shows a small part of a contour map, that has been scanned. The samples were extracted, their elevation values were assigned and a TIN model was created. The contours were densely sampled so the crust reconstructs their shape and can also be used to assign them elevation values attaching a specified elevation value for crust vertices representing each contour. Figure 6.32b shows a perspective view of the resulting terrain model with the contour lines draped over the terrain. It is seen in many places in Figure 6.32b, that instead of ridges, valleys or summits, flat terraces have been created ("flat triangles"). These flat features are formed by triangles having all three vertices at the same elevation. The problem of flat triangles is a very well know problem of TIN models created from contours and can be solved by insertion of the skeleton point of each flat triangle, which guarantees that it is replaced by new triangles with the skeleton point as a vertex.

[1]This section is based on Dakowicz and Gold (2003).

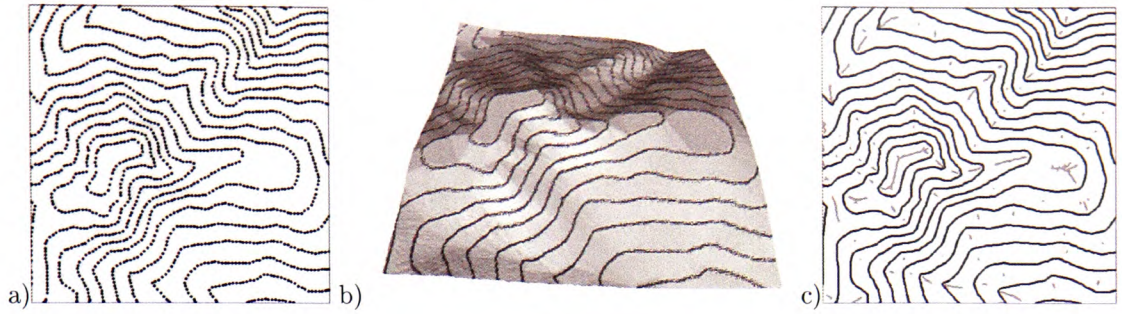


Figure 6.32: Contour data. a) Samples of contour lines. b) A TIN generated from the samples with flat triangles visible. c) The crust and skeleton branches. (from Dakowicz and Gold (2003))

However, these skeleton nodes have no elevation information, so the challenge is to assign them meaningful elevation values. The technique of Thibault and Gold (2000) uses the concept of Blum (1967) of height as a function of distance from the curve or polygon boundary. This is illustrated in Figure 6.33a, where points on a simple closed curve are used to generate the crust and skeleton, and the circumcentres of the skeleton points are given a height above the previous contour level equal to the circumradius. The resulting interpolated model is shown in Figure 6.33b. This model is based on the idea that all slopes are identical, and thus the radius is proportional to the height of the skeleton point. Of course, in the case of a real summit, the slope would initially be unknown, and would be estimated using circumradii from the next contour level down – see Thibault and Gold (2000). This technique can be also used in the case of ridges and valleys. An alternative method is to assume a constant slope down the valley or up the ridge (see Dakowicz and Gold (2003) for details).

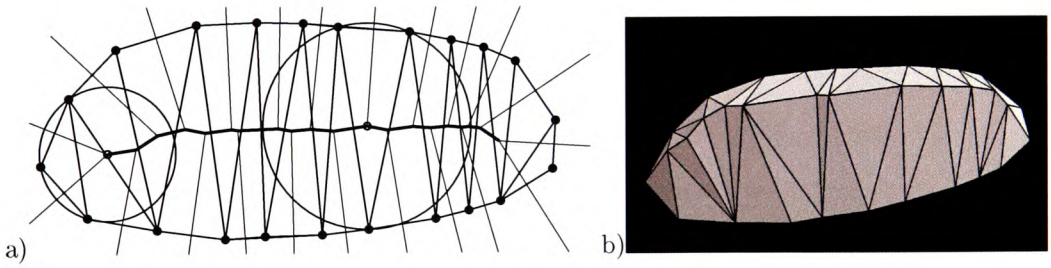


Figure 6.33: The triangulation of a summit. a) The skeleton and circumcentres. b) A perspective view of the elevation model after adding skeleton vertices with assigned height values. (from Dakowicz and Gold (2003))

Figure 6.34 shows the improved model when estimated skeleton points are added. All flat triangles are removed and ridges, valleys and the summit are reconstructed.

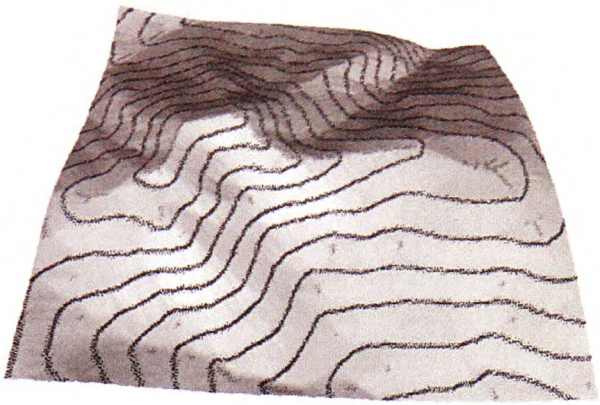


Figure 6.34: The enriched terrain model. (from Dakowicz and Gold (2003))

Summing all the above information, a general approach for the common problem of deriving DEM surfaces from simple TIN models generated from contours can be divided into five main operations:

1. Generate skeleton points along the ridges, valleys, pits, summits and passes using the method of Thibault and Gold (2000).
2. Assign elevations to these skeleton points by the methods described here, or other suitable techniques.
3. Eliminate flat triangles by inserting these skeleton points into the original TIN.

Additionally, if a grid model is preferred the resulting TIN model needs to be interpolated. To ensure a good quality of the grid model two extra steps are necessary:

1. Estimate slope information at each data point by any appropriate technique.
2. Perform weighted-average interpolation using the previously estimated slope information. Avoid methods, such as the gravity model, with exponentially large close-range weightings, and avoid neighbour selection techniques which require user-specified parameters, such as counting-circle radius.

The disadvantage of the resulting grid surface is a slight discontinuity of the slope along the contour lines. An improved smoothing function that guarantees both slope continuity and minimum curvature might solve that problem.

6.5.3 Interpolation[2]

Longley et al. (2001) defines spatial interpolation as a "*process of intelligent guesswork, in which the investigator (and the GIS) attempt to make a reasonable estimate of the value of a field at places where the field has not actually been measured*". This operation can only be used in field models and it used to estimate the value of the attribute at a specified location, where this value is unknown. Field models usually exhibit smooth variation of the attribute. The value is calculated using a specified set of existing observations - their selection is the crucial component of interpolation.

[2]This section is based on Dakowicz and Gold (2003).

Interpolation is used to create discretised continuous surfaces from sample points or to re-sample grids to different resolutions (Burrough and McDonnell (1998)).

The VD/DT can be used as a base for several weighted-average methods. The data used for the comparison consists of a set of enriched contours, shown in Figure 6.32c. In general, the weighted average functions f used to interpolate a point x having k neighbours can be formulated as follows: $f(x) = \frac{\sum_{i=1}^{k} w_i(x) * z_i}{\sum_{i=1}^{k} z_i}$, where $w_i(x)$ is the weight of each neighbour of x and z_i is the attribute of x.

There are three components of a weighted-average interpolation method: the weighting process used, the set of neighbours used to obtain the average and the elevation function being averaged (the data point elevation alone or a plane through the data point incorporating slope information as well).

One simple weighted-average model is triangle-based interpolation in which a linear interpolation is performed within every triangle T. The areas a of three triangles sharing the interpolated point location inside T are used as weights. Each area is used together with the attribute z of the opposite vertex of T. The result of applying this method on the TIN generated from the enriched data in Figure 6.32c is shown in Figure 6.35a. This method was used in the previous paragraph to illustrate a problem of flat triangles (Figure 6.32b) and to show the result of simple TIN model enrichment (Figure 6.34).

Another popular weighted average model is the traditional "nearest n-points" method, also called "the gravity method" or "the inverse distance weighted" interpolation (IDW). In this case the weighting of each data point used is inversely proportional to the square of the distance from the data point to the grid node being estimated, although other exponents have been used. There is no obvious set of data points to use, so one of a variety of forms of the "counting circle" is used for selecting them. Figure 6.35b shows the resulting surface for a radius of about a quarter of the map. Data points form bumps or hollows. If the radius is reduced there may be holes in the surface where no data is found within the circle. If the radius is increased the surface becomes somewhat flattened, but the bumps remain. The result depends on the radius, and other selection properties, being used. Clearly, in addition, estimates of slope would be very poor, and very variable.

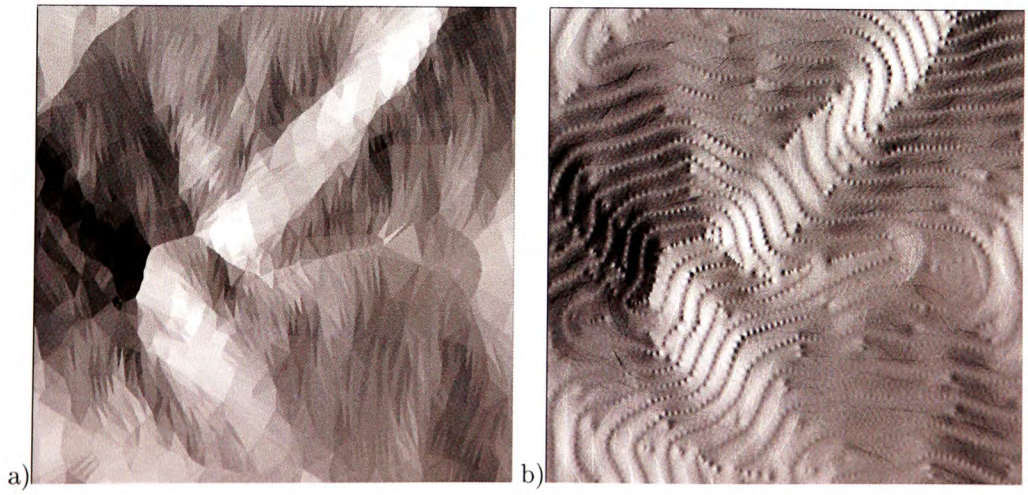


Figure 6.35: Simple interpolation methods. a) The triangle-based interpolation. b) The gravity model. (from Dakowicz and Gold (2003))

The Sibson method is also called the "area-stealing" or "natural neighbour" interpolation ((Gold

(1989), Sibson (1980), Watson and Philip (1987)). It is based on the idea of measuring the areas that a dummy point inserted at the interpolated location would "steal" from its neighbours and using them as the weights for the weighted-average. Those neighbours (called the natural neighbours) are well defined, since the insertion of a point in the VD produces a unique result. In order to obtain the stolen areas the insertion and deletion sequence can be replaced by mimicking the insertion, without modifying the triangulation (Watson (1992)). The deletion operation then is not required, which increases the speed of the process.

Figure 6.36 shows a sample data set and the neighbour selection for the same point in both the gravity (Figure 6.36a) and Sibson method (Figure 6.36b). In the Sibson method the natural neighbour selection results in a reasonable set of neighbours, but the circle used in the gravity method may not select a sufficient number of neighbours to produce a valid elevation value for the interpolated point. The Sibson method is particularly appropriate for poor data distributions as the number of neighbours used is well defined. In the gravity model, when the data distribution is highly anisotropic, there is considerable difficulty in finding a valid counting circle radius.

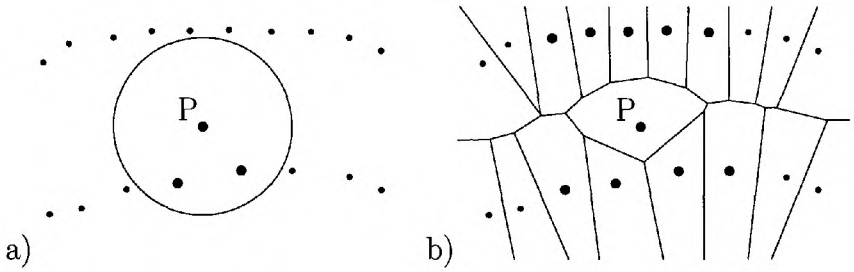


Figure 6.36: The neighbour selection. a) Using a counting circle. b) Using Voronoi neighbours. (after Dakowicz and Gold (2003))

Figure 6.37a shows the results of using Sibson interpolation. The surface behaves well - it fits the original data and is smooth in areas between data points, but is angular at ridges and valleys. Indeed, slopes are discontinuous at all data points (Sibson (1980)).

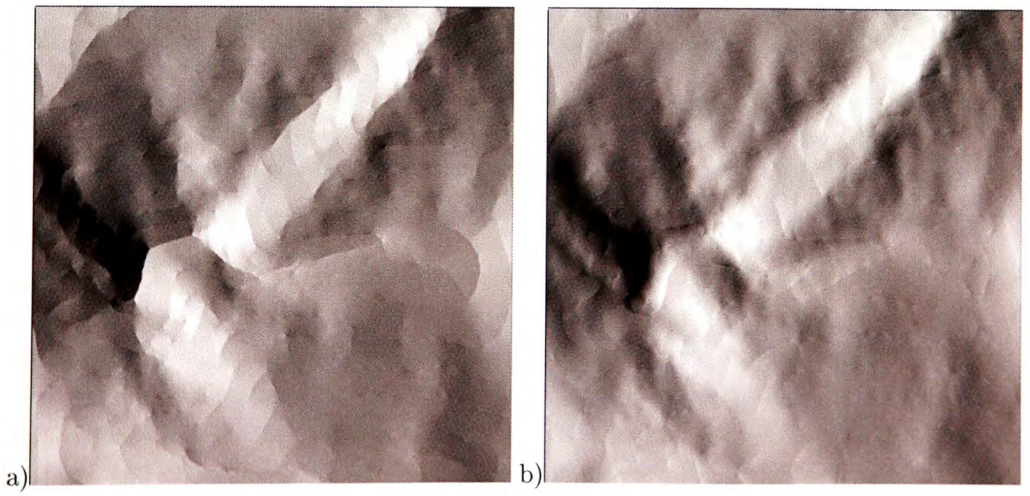


Figure 6.37: Sibson interpolation. a) Without slope information. b) Using slopes. (from Dakowicz and Gold (2003))

In real applications, however, accuracy of slope of the generated surface is often more important

than accuracy of elevation – for example in runoff modelling and erosion. Clearly an assumption of zero slope at each data point, as above, is inappropriate. However, in the weighted-average operation the height of a neighbouring data point can be replaced by the value of a function defined at that data point – probably a planar function involving the data point height and local slopes. At any interpolated location the neighbouring points are found and their planar functions are evaluated for the x,y position. These z estimates are then weighted and averaged according to the interpolation method used.

Figure 6.37b shows the result of using Sibson interpolation with data point slopes. The result provides us with a surprisingly realistic model of the surface - that is, one that conforms well to our subjective interpretation of what a real landscape should look like. However, there are still slight breaks in slope at contour lines.

Adding slopes to the simple TIN model (i.e. using the position in the triangle to provide the weights) produced results (Figure 6.38a) that were almost as good as the Sibson method when the sample points were closely spaced along the contours. However, the Sibson method is much superior for sparser data, or where the points do not form contour lines. The gravity model does not provide particularly good slope estimates (Figure 6.38b), but even here including the data point slope function produces a significant improvement.

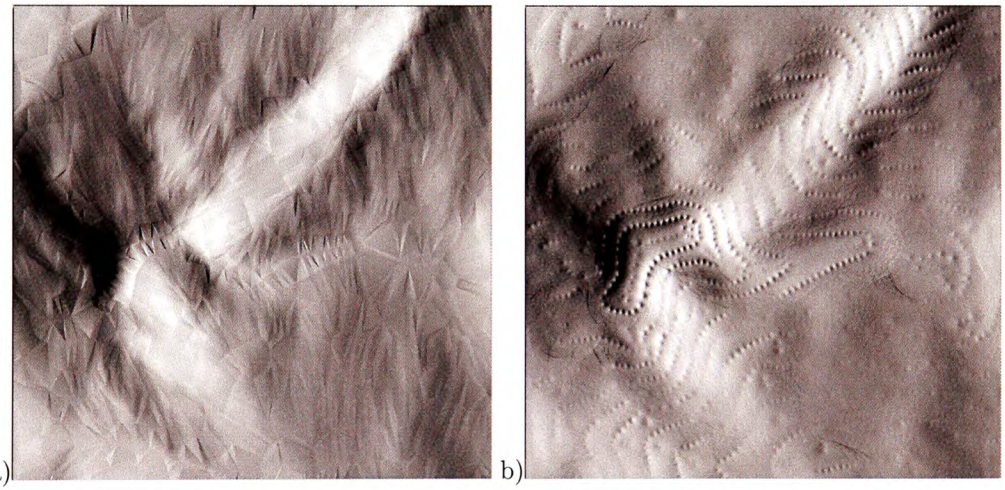


Figure 6.38: Adding slopes at data points. a) The triangle-based interpolation. b) The gravity interpolation. (from Dakowicz and Gold (2003))

6.5.4 Terrain Modification[3]

In the real world landscape modification is usually achieved by earth "sculpting" with heavy equipment. Before starting the real work, many times there is a need to simulate and evaluate the process first. In the digital design process the engineer needs to be able to visualize the effect of his proposed terrain modification, and adjust his design accordingly. The main effects under consideration are the visual effect, as in landscape architecture (Ervin and Hasbrouck (2001)), the desired surface form and the volume moved, as in road construction, and the resulting slope changes, erosion potential, landslide potential, runoff and water collection and hydrograph modelling for dam construction and catchment modelling. The ability to model the landscape interactively, view

[3]This section is based on Dakowicz and Gold (2005b) and Dakowicz and Gold (2005a).

the effects, and then adjust the model again as desired, has great value in various aspects of civil engineering and construction.

The terrain modification process requires two objects: the terrain to be modified and the tool to modify the terrain. There are several possible representations of the cutting tool. Nienhuys and van der Stappen (2004) used a point-shaped scalpel that moved an active node through a static mesh and the area around the active node was re-meshed during the movement. Bruyns et al. (2002) in their real-time surgical simulation environment used a scalpel, which is a single cutting surface. Additionally, they proposed a method of scissors cutting, where two edges were used to cut a surface. They also produced a survey of mesh-cutting techniques, addressing several major issues in the mesh modification process. Those included the definition of the cut path, the way the cutting operation is implemented (removal or re-meshing intersected primitives), the number of new primitives created during the re-meshing process, when exactly the re-meshing operation is performed and the representation of the cutting tool.

Apart from the terrain (Gold and Dakowicz (2000); Dakowicz and Gold (2002a)) and the cutting tool, a 3D environment allowing display and manipulation of them simultaneously is necessary. The "GeoScene" (Gold et al. (2005)) implemented using in OpenGL graphic library has been used as the core of the 3D environment.

The TIN modification process is based on the idea of slicing objects with a knife. The object is assumed to be transparent, so the cutting tool can be freely navigated in the scene, intersecting the object when required. The knife, which can be called a "cutting plane", is modelled in 3D as a rectangle defined by four corners with x, y and z coordinates (Figure 6.39). There are two main attributes associated with the cutting process – the knife's size and position. The size of the knife determines the range of changes of the surface – a small knife can be used for small shape adjustments and a large knife can rapidly change big areas of the terrain. All four edges of the knife are treated as cutting edges. The position of the knife in the coordinate system of the terrain object is defined by the coordinates of the four corners and allows computing of intersection points between the knife and the surface. Using the 3D interface the user is able to adapt the knife's size to the scale of the modified surface features, and to set its position precisely in the model at the desired position of the cut.

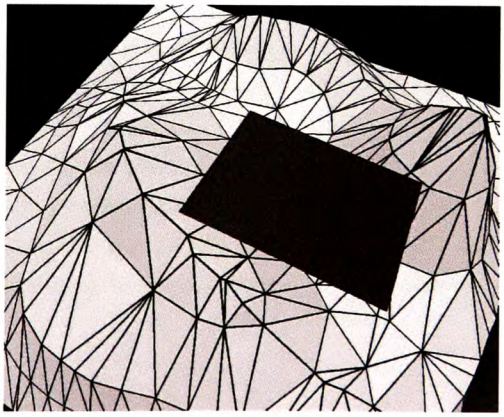


Figure 6.39: The knife above the terrain. (from Dakowicz and Gold (2005b))

There are two possible operations – a removal of a volume above the knife's surface and filling up the space below the knife. The most common example of the removing operation (decreasing

the volume of the model) is a simulation of a road construction when a road is being carved on the side of a hill. The filling up operation (increasing the volume of the model) can be used for building or dam construction.

The cutting technique is a rapid process performed after positioning the knife in the scene. Basically, the process consists of three main operations – a computation of intersection points between the knife and the terrain, a removal of specified existing nodes and an insertion of intersection points. In some cases usage of additional knives is necessary – for example one additional knife is needed to remove a part of a summit, and two additional knives are needed to construct a dam in a valley. Intersection points are computed testing edges of each TIN triangle against the knife's rectangle. For each intersected triangle its intersection points are stored in a list as a pair that has to be connected by edges in the final model.

The algorithm is as follows:

1. Computation of intersection points using the main knife, on the surface of the terrain and along edges of the knife if necessary. Storing points as pairs that have to be preserved in the final model.

2. Testing whether the main knife's corners have to be included in the final model.

3. Addition of additional knives "touching" the main knife and computation of intersection points with the surface of the terrain.

4. Removal of existing data points above or below all knives, temporarily deforming the shape of the model.

5. Insertion of the knife's corner points if necessary.

6. Insertion of intersection points along common edges of knives in case of using additional knives if necessary, preserving connectivity between pairs of intersection points.

7. Insertion of intersection points on the surface of terrain preserving connectivity of pairs.

Pairs of intersection points have to be preserved in the final model to keep the shape of the surface on the edges of the cut. They are preserved using the idea of the Conforming Delaunay Triangulation, by insertion of intermediate points between two existing nodes until those nodes are connected with edges of triangles. See Dakowicz and Gold (2005b) for more details about this process.

Figure 6.40 presents steps of the process of removing a part of a hill, where one additional cutting plane is involved in the process to make the additional almost vertical cut. This additional plane cannot be completely vertical due to the assumptions of 2D Delaunay triangulations, where two points cannot exist at the same location, even if they have different elevation values.

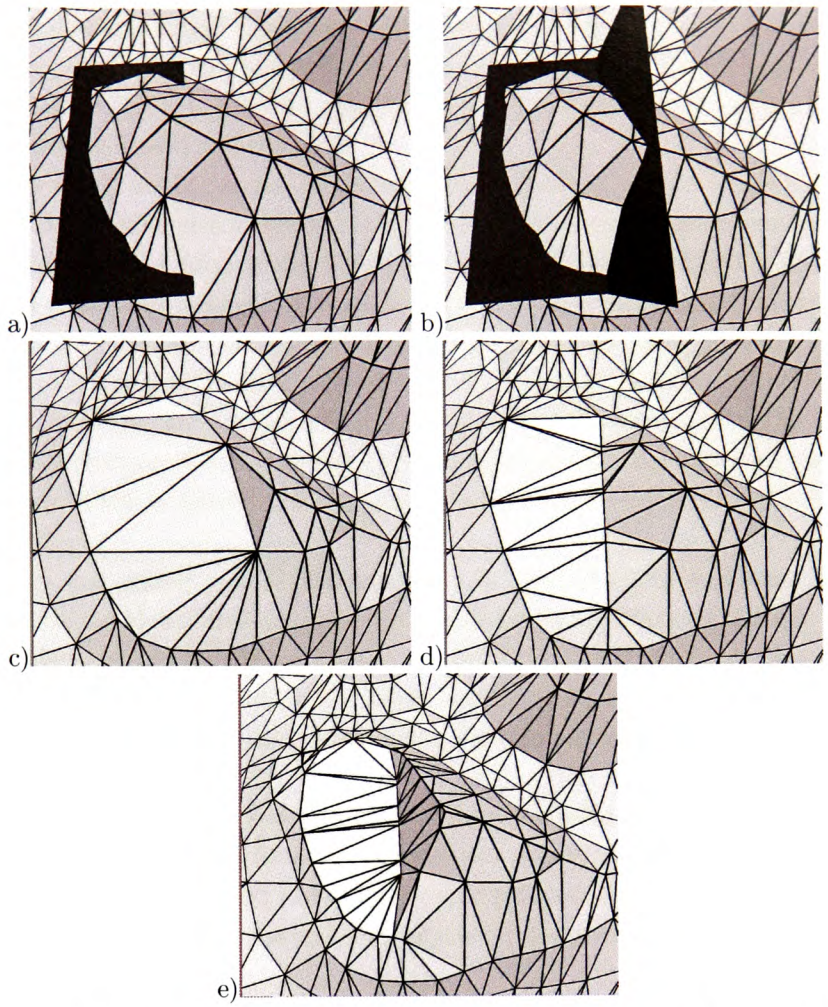


Figure 6.40: Removing a part of a summit. a) The positioning of the knife. b) The additional knife. c) The surface after removing points above knives. d) The surface after insertion of points along the common edge of two knives. e) The final surface after insertion of intersection points. (from Dakowicz and Gold (2005b))

The knife is positioned horizontally, with the cutting edge inside the hill (Figure 6.40a) – then all intersection points are on the side of the hill. Then a second knife (almost vertical) is added at the cutting edge of the first knife (Figure 6.40b), with intersection points on the top of the hill. Having computed intersection points for both knives, all points inside the area defined by a closed curve formed by two sets of intersection points can be removed (Figure 6.40c). The next step is to create a straight edge along the cutting edge of the first knife, where two knives meet (Figure 6.40d). Finally, intersection points on the surface of the terrain can be inserted, producing two flat areas on the top of the hill - one horizontal and the other almost vertical (Figure 6.40e).

6.5.5 Runoff Modelling [4]

Systems for digital runoff simulation consist of two main components: the digital terrain representation and the hydrologic processes governing overland water flow. The two most popular digital representations of terrain are the regular grid and the triangular irregular network (TIN),

[4]This section is based on Dakowicz and Gold (2007a) and Dakowicz and Gold (2007b).

both suitable for dynamic runoff simulation. The availability and simplicity of grid models made them the most popular representation and base for various hydrological modelling processes (Burrough and McDonnell (1998)). However they have many disadvantages, including flow directions restricted to 45 degree direction increments.

The two most widely used formulations for water flow modelling are The Finite Element Method (FEM) and the Finite Difference Method (FDM). FEM methods use irregular meshes, FDM methods are often grid based. A form of the FDM, the Integrated Finite Difference Method (IFDM), is based on non-structured meshes. A manual method of irregular cells construction originally suggested by MacNeal (1953) was fully defined by Narasimhan and Witherspoon (1976), and was automated using the Voronoi diagram by Lardin (1999), showing that an iterative finite difference scheme could be developed for Voronoi cells ("buckets" to hold the water), using the dual Delaunay triangulation. The volume of water moved between adjacent cells depended on the gradients of the triangle edges. Flow is distributed irregularly, based on the distribution of cells adjacent to the processed cell.

The efficiency and stability of IFDM simulation is determined by the shape of cells. The method is based on the idea of moving water volumes between neighbouring Voronoi cells with the amounts determined mostly by the gradient of the cells, the width of the common edge and the size of cells. To make the process stable it is recommended to regularize the shape of the cells (see the differences in sizes of the cells in (Figure 6.41a) by thickening the mesh data with additional nodes.

There are two steps - adding a flow rectangular frame enclosing all the data (Figure 6.41b) and thickening the data inside the flow frame using a random pattern of points with a guaranteed minimum spacing (a relatively large disk radius assigned to points and used for the collision detection) and elevation values assigned using a specified interpolation method, preferably Sibson's (Sibson (1982); Gold (1989)).

This leads to a fairly uniform distribution of the triangulation nodes (Figure 6.42a). Figure 6.42b shows the 3D view of the resulting TIN model with the original contour lines draped over the terrain. Figure 6.42c shows the 3D view of the Voronoi cells of the same model.

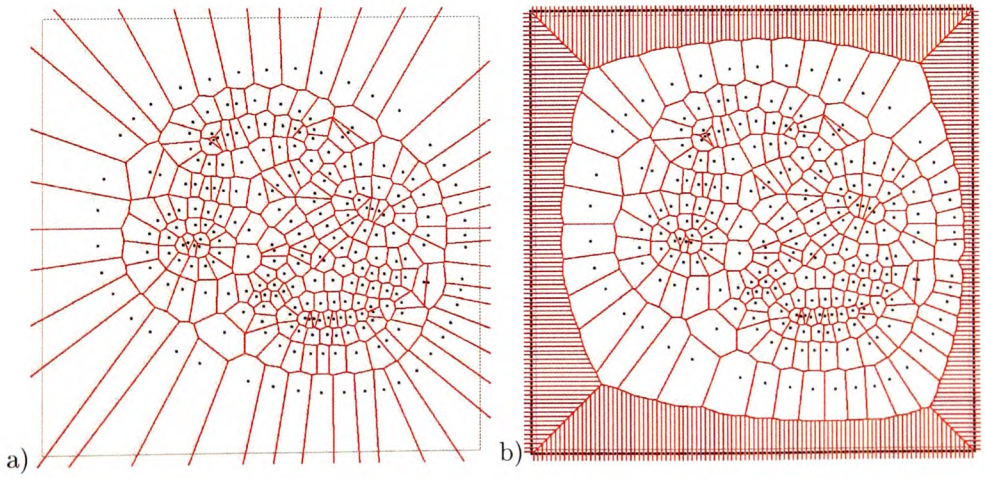


Figure 6.41: The Voronoi Diagram of the input data. a) Before adding the flow frame. b) After adding the flow frame. (Figure b after Dakowicz and Gold (2007a))

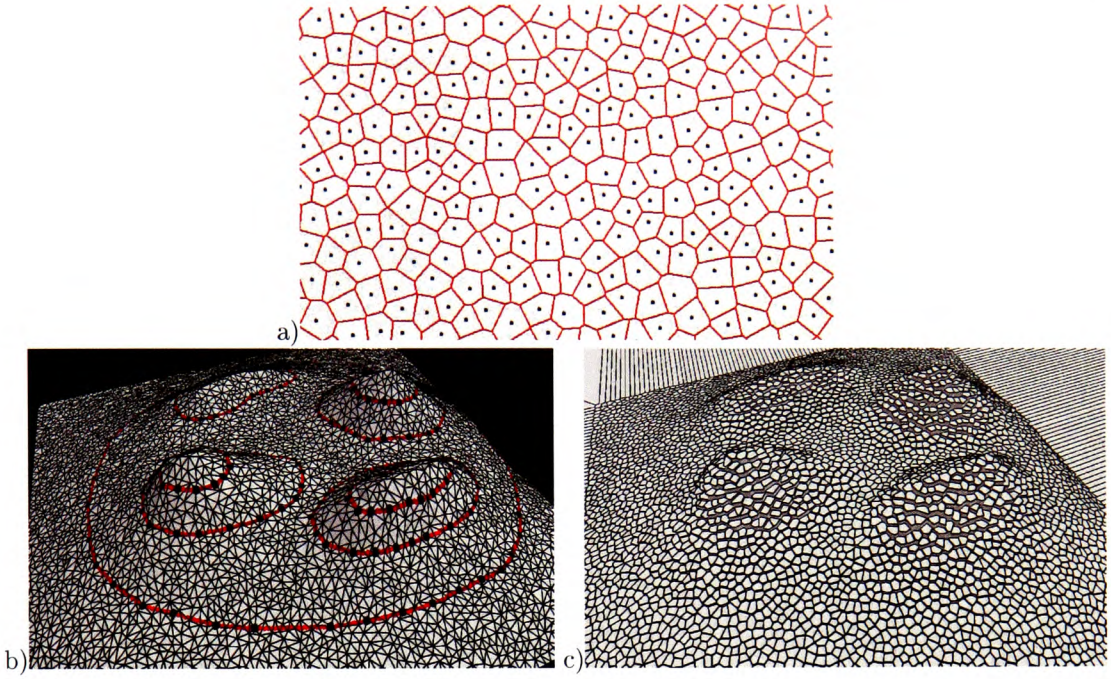


Figure 6.42: The enhanced TIN. a) The enriched Voronoi diagram. b) The 3D view of the Delaunay triangulation with superimposed contour lines. c) The 3D view of the Voronoi cells. (from Dakowicz and Gold (2007a))

The runoff simulation process itself consists of two main steps – the initialization and the iterative processing of the model. In the real world the initialization step corresponds to rainfall over a specified area, so at a specified moment a certain amount of water covers the surface. Voronoi cells are treated like "buckets" and water is "poured" into each of them (Figure 6.43a) by scanning the whole map and assigning a specified water height value to each node.

The flow simulation itself is performed iteratively for the whole map for a number of iterations specified by the user. Firstly the water volumes to be moved between adjacent cells are computed. All edges of the mesh are processed and each edge is used once. One cell is a source of water for all the neighbouring cells with lower elevation and at the same time it receives water from the higher neighbouring cells. The elevation values are combined with the water level in the cell – that solves the "pit" problem in runoff modelling

A cell with a higher elevation at one end of the processed edge is set as the source and a specified amount of water flows from this cell to the cell at the other end (see Dakowicz and Gold (2007a) for more details). The volume of this flow depends on the source cell area, the gradient of those two neighbouring cells and the length of the common Voronoi edge (see Narasimhan and Witherspoon (1976) for the details of the flow formulation). Gradients are computed using the distance between two nodes and their elevation values combined with the level of water. Two other factors affecting the volume are the time interval between iterations and the resistance of the surface at each node location. The time parameter is set before starting the process. The resistance depends on the surface material and it can be different in various parts of the map. In the preliminary model the same resistance value is set for the whole map, but it changes according to the current level of the water in the cell – an increase of the water level decreases the resistance, which means that deep water flows faster than shallow. In practice, before applying the resistance parameter it is tested if the water level in the cell is higher than the specified threshold value – in such case a predefined

smaller resistance value is used.

Figure 6.43 presents the flow process for an imaginary set of hills. Firstly, the model is initialized by adding the same amount of water to each cell (Figure 6.43a), which simulates rainfall. Then the flow process is iterated (Figure 6.43b-d) and water flows from the hills downwards.

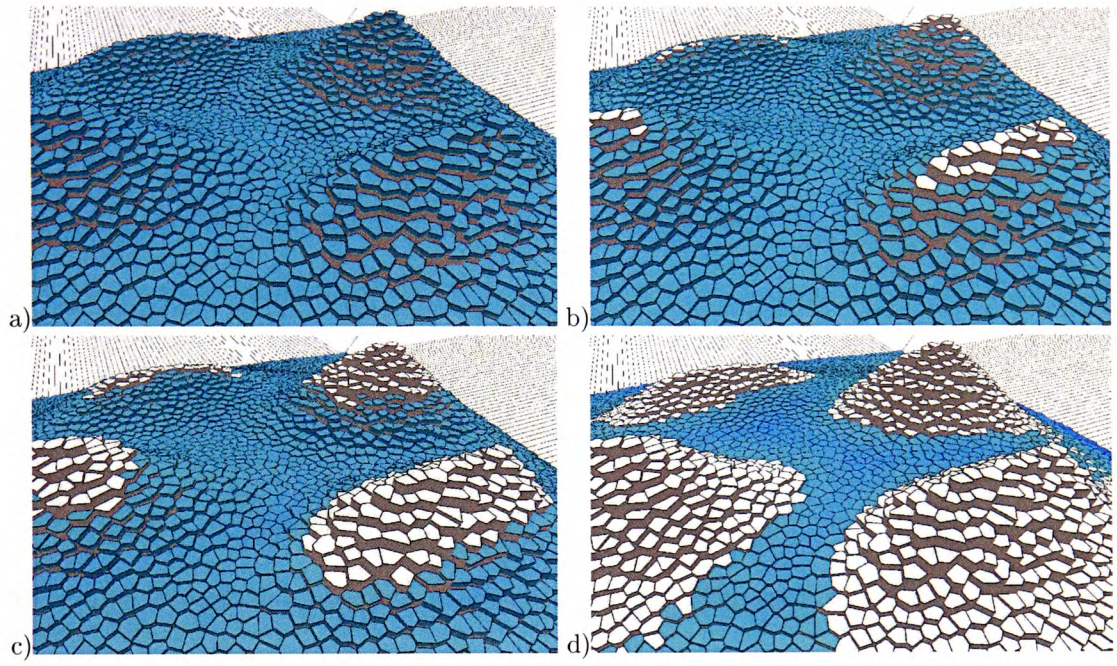


Figure 6.43: The 3D view of the flow process. a) The initial model. b-d) The model after various numbers of iterations. (from Dakowicz and Gold (2007a))

A visual inspection shows a reasonable distribution of the water in the model at each step of the process. Thanks to the good quality of terrain models there are no artificial places of water accumulation. Water flows in various directions (determined by the number of neighbours of each cell), unlike in grid models where it is restricted to eight directions only. Water levels have been monitored at specified locations and the shape of the hydrographs conforms approximately to the theoretical assumptions. As expected, the flow volume peaks soon after the rainfall (the initialization) and decreases steadily thereafter. The main drawback of the method is its speed - mostly due to the complexity of terrain models.

6.6 Operations for Multi Layered Models

All previously described applications were these using a single map. However, the strength of GIS lies in the possibility of working with more than one map, performing various kinds of queries and combining maps according to needs. The two traditional GIS operation involving more than one map are "point in polygon" and "polygon overlay". Using the VD based layers these operations can be performed on all types of input maps using the same mechanisms.

6.6.1 Background

One of the most important features of GIS is the ability to work with different thematic maps of the same area (layers), place one onto another and use them for analysis or producing new maps

(O'Sullivan and Unwin (2002)). The procedure depends on the location of each feature, so it is required that layers use a common coordinate system (Bolstad (2003)). This is shown in Figure 6.44 where a map of green fields is put on top of a map of the road network of the same area to identify parts of roads passing through green areas.

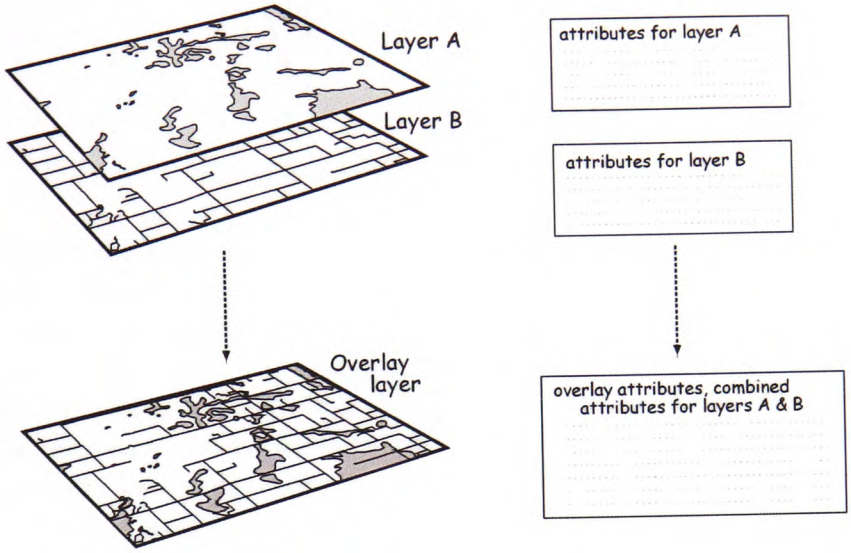


Figure 6.44: Spatial data overlay combines both the coordinate and attribute information (from Bolstad (2003)).

The two most popular traditional operations involving two maps are "point in polygon" and "polygon overlay". The point in polygon operation, often called inclusion, in the simplest form is used to test whether a point lies inside or outside a polygon. In a more complex form, when there are many points and polygons, it assigns points to polygons enclosing them (Longley et al. (2001)). This operation is called using a more general name "merge" in this research, as using the VD structure it can be also employed to identify objects in maps different than polygonal, for example it can locate parts of the network nearest to specified locations.

The polygon overlay, called "merge" in this research, has different meanings for fields and objects (Goodchild (2002)). For discrete objects it involves testing if two objects intersect and computing the new area. This can produce no result, when there are no overlapping objects, or one or more objects when they do overlap. The result can be interpreted in several ways, depending on the application. Usually algebraic operations are used, such as "AND" (intersection) producing the common area of the input polygons, "OR" (union) resulting in a sum of both areas or "NOT" returning parts of one area not overlapped by the other one. Figure 6.45 shows two polygons and the result of the overlay function producing six new polygons. The region a is a result of "AND" operation while "OR" operation returns all six regions.

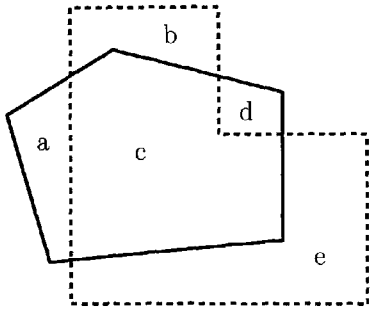


Figure 6.45: Two overlapping polygons.

In case of fields the result is a combination of two input maps. The new map is divided into areas having boundaries determined by intersections of areas that occur between two processed maps. Each area has two attributes taken from each map. All boundaries are preserved, but they are broken into smaller pieces by the intersection points. The operation is performed differently for grid and vector representations. For grids (having the same resolution), after rendering them into the same coordinate system the overlay procedure means performing Boolean or arithmetic operations on attributes of pairs of cells. The overlay operation for vector models is more complex. The process consists of geometric and attribute phases (Chrisman (2002)). In the geometric phase intersections between input lineworks are discovered and a new topological structure is created. All polygons are labelled with unique identifiers linking them with the source attribute tables. In the attribute phase results are produced.

6.6.2 Query and Merge Using Voronoi Diagrams

As was mentioned before, the traditional GIS usually handles four main types of spatial data categories: discrete objects (points, lines and mobile points), networks, polygons and surfaces. All these can be converted to Voronoi diagram based layers and used to perform "query" and "merge" operations. Discrete point layers and surfaces are just ordinary point VDs/DTs. Their physical representation is the same, the only difference is the way they are interpreted. A triangulation of points may be considered just a network of triangles connecting points or a surface defined by a particular interpolation method such as linear interpolation or Sibson's. Similarly the discrete and space exhausting polygons - they are just LSVDs or CDTs with polygons drawn as line segments or constrained edges. Also lines and networks can be represented as one of these two structures, since their segments can be incorporated into the VD/DT as line segments or constrained edges. So we can notice that all spatial information can be represented either by the ordinary point VD/DT or the LSVD/CDT. However, all these diagrams share the same structure and are managed using the same set of operators so we can say that all common spatial data can be represented in GIS using the VD/DT model.

The "query" operation requires that one of the layers is a point map consisting of one or more points. It does not involve merging two layers, but is performed to obtain information from another layer at the specified location(s) to find what is present there, what is the elevation at a specified point or what is the area of a selected polygon. The overlaying upper layer contains points only, while the overlayed bottom layer can be of any type. For the VD based layers this means locating the features (points, line segments or constrained edges) in the bottom layer for each of the points from the upper layer. This identifies the nearest point or a set of points, network segment or

polygon in the bottom layer and the result can be interpreted in any desired way.

If the bottom layer is an ordinary VD of points then the query is equivalent to interpolation, either nearest neighbour, linear or Sibson's. If the bottom layer is a LSVD (or the CDT) representing lines, networks or polygons (discrete or space exhausting) then the query locates the nearest segment of the diagram for each point from the upper layer. The result can be interpreted as the nearest road section for each mailbox, or the parcels containing locations of electric poles, for example.

The "merge" operation (the traditional overlay) on layers represented by Voronoi diagrams is used to combine two maps. Technically any combination of the input layers can be merged because they are just Voronoi diagram, but not every combination is meaningful in GIS. The only practical limitation is that LSVDs and CDTs cannot be merged as the intersection of line segments and constrained edges is not handled by the algorithm.

The operation consists of two parts. The algorithmic part loads one data set into another one, snapping lines or points together whenever collisions, or close collisions, occur (so sliver polygons are not introduced). The second part is the interpretation of the resulting diagram dependant on the knowledge about the input maps, for example whether line segments represent roads, rivers or polygon boundaries.

Merging two point based VDs, which combines two sets of points, can merge locations of electric and telephone poles or enhance a surface with additional elevation points.

Merging point VDs with LSVDs representing polygons can be used to produce maps showing the distribution of occurrences of events in a specified area as illustrated in Figure 6.46. Figure 6.46a shows a map of boundaries of the electoral wards of Cardiff. These boundaries can be used to calculate a LSVD (Figure 6.46b). Figure 6.46c shows the distribution of fire accidents in Cardiff. Figure 6.46d shows these fire locations inserted into the LSVD map of Cardiff, so each fire location has its own Voronoi cell and neighbouring line segments are readily available. The same is true for lines and networks.

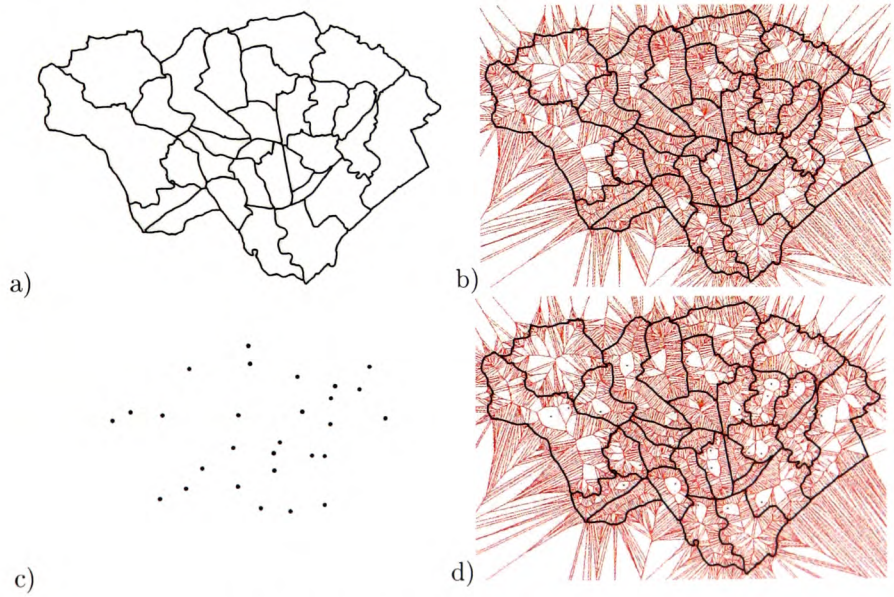


Figure 6.46: Fire cases in Cardiff illustrating "point in polygon" operation. a) Boundaries of electoral wards. b) The LSVD of the wards. c) Fire occurrences. d) Fire occurrences inserted into the LSVD.

Merging two networks can be use to join them into one network system. A common example is merging road (Figure 6.47a) and rail (Figure 6.47b) transportation networks into one transportation network map (Figure 6.47c). The resulting LSVD (Figure 6.47d) can be used for various analyses and operations.

Figure 6.47: Merging two networks. a) Road network. b) Railway network. c) Overlay of two networks. d) The LSVD of merged networks.

Another possible merging operation is placing a network map on top of a polygonal map, which can be used to identify sections of the network in each polygon or just to join two maps. Figure 6.48 shows the process of overlaying polygonal and network maps, depicting building outlines in a specified area (Figure 6.48a) and the road network of this area (Figure 6.48c). Figure 6.48b and Figure 6.48d show the LSVD created from both maps. Figure 6.48e shows both maps merged into one layer and Figure 6.48f shows the LSVD created from the merged data by loading one map into another.

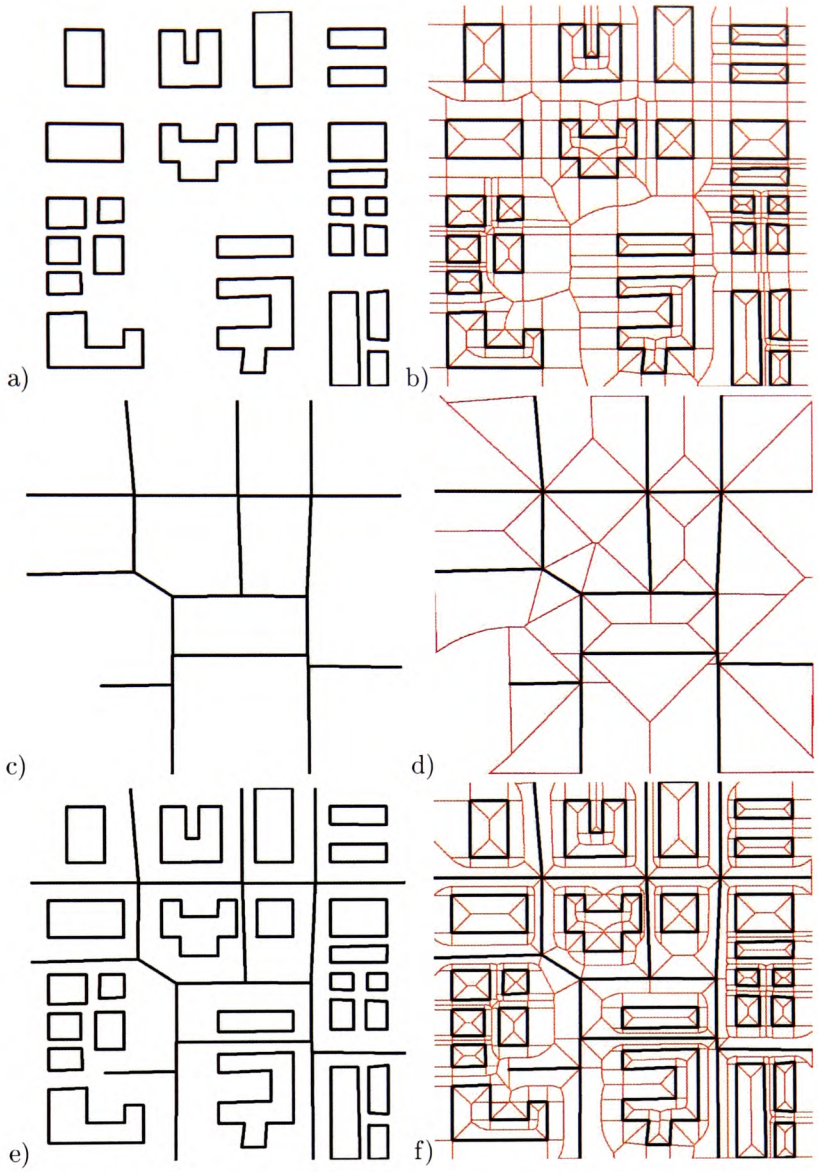


Figure 6.48: Overlay of a polygon and network map. a) A polygonal map of buildings. b) The LSVD of the polygonal map. c) A road network. d) The LSVD of the network map. e) Two maps merged. f) The LSVD of merged maps.

Figure 6.49 presents merging two polygonal maps. One (shown in Figure 6.48a) contains outlines of buildings and the second (Figure 6.49a) presents buffer zones of the road network map from Figure 6.48c. These buffer zones mark the areas of the map that are within a specified distance from the road and can be used to detect if proposed road widenings collide with buildings present in that area. Figure 6.49c shows these two maps merged into one layer and Figure 6.49d shows the LSVD of these two merged maps. It is seen that some buildings overlap these buffer zones, so such widening of is not possible.

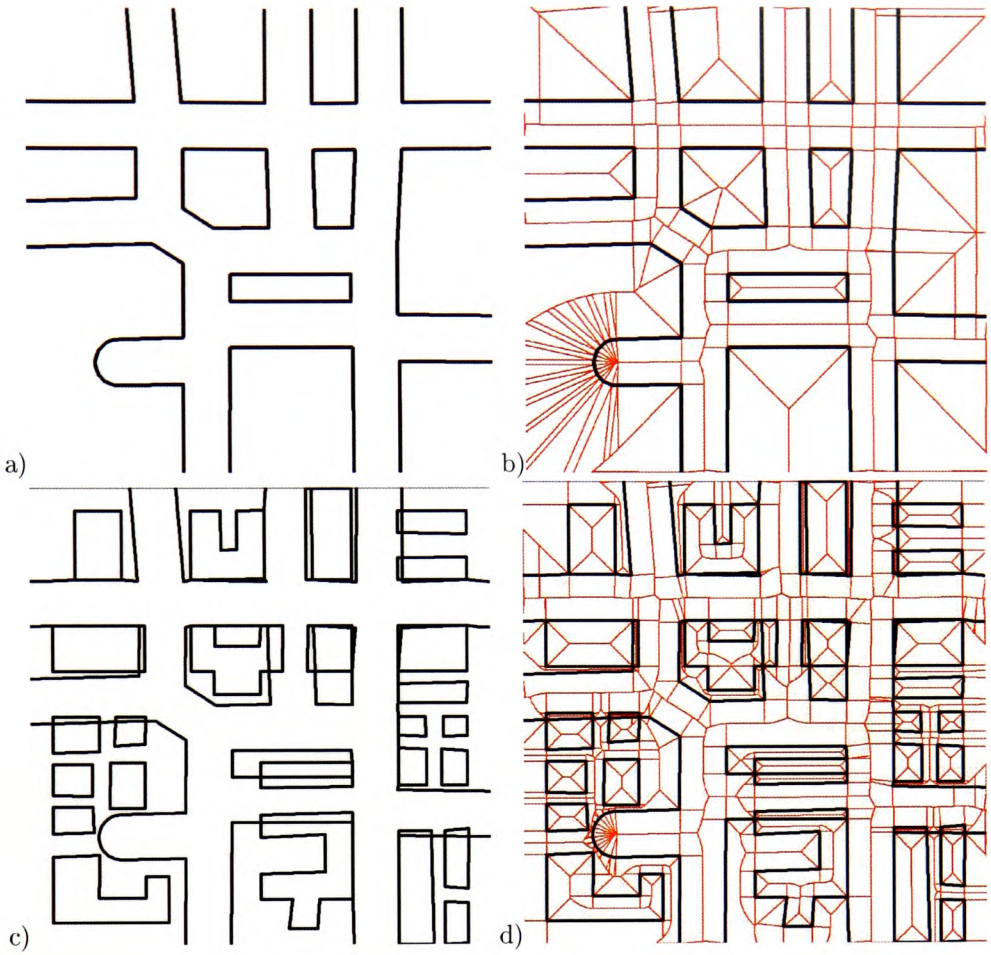


Figure 6.49: Overlay of two polygonal maps. a) Buffer zones of the road network from Figure 6.48c. b) The LSVD of the buffer zone map. c) The polygonal map of buildings from 6.48a and the buffer zones merged into one layer. d) The LSVD of two merged layers.

Figure 6.50 presents a workflow of potential park sites identification (after Bolstad (2003)), which summarizes usages of typical GIS map operations discussed in this chapter. The suitable sites are the areas located near lakes and roads which are not wetlands. Three input maps containing lakes (discrete objects), roads (network) and hydric status (polygon map) are used as the input data. Various spatial operations are applied to these maps, including buffering, reclassification and overlay. The result is one suitability map that can be used to identify owners, or help in park site selection. All these steps can be performed using the techniques described here.

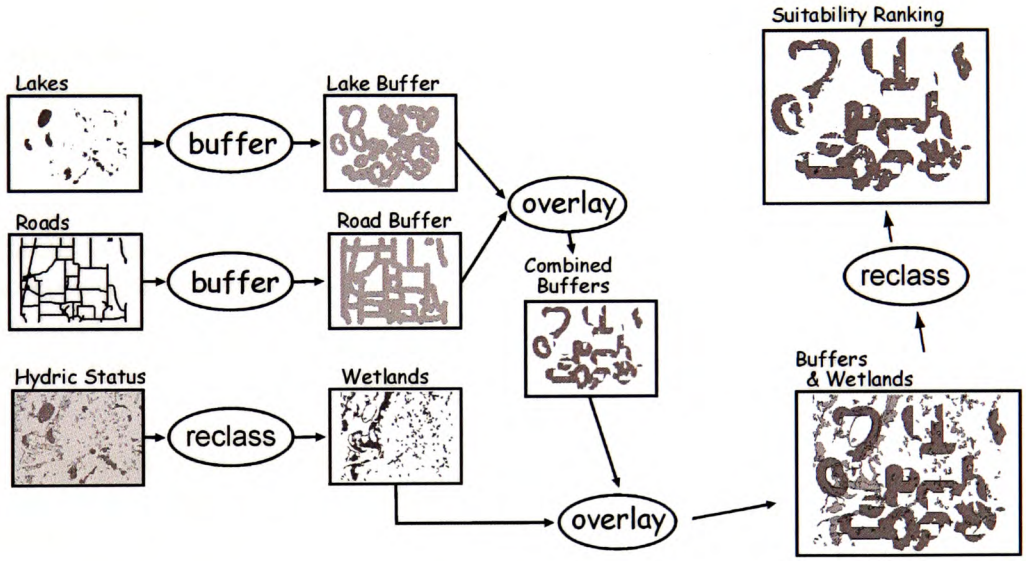


Figure 6.50: A workflow of identification of potential park sites using three input maps (from Bolstad (2003)).

Chapter 7

Conclusions

Many researchers have pointed out the importance of the data structure selection to manage geographical phenomena in GIS, as it determines the range of available functions and analyses (Theobald (2001)). Conceptually these phenomena can be of a discrete object or field type. Discrete object models populate the space with entities and empty space can exist between them. Fields cover the whole map with the attribute values. GIS traditionally organizes data into layers and uses different types of representations for different types of data. Separate building are stored as unconnected polygonal objects, locations of events as unconnected points, roads as lines connected into networks, postcode areas as polygons, topologically connected or not, and surfaces as regular grids or triangulations. Although it is possible to operate on different layers simultaneously there is no consistent method applicable to all types of data, as different methods are required for storage, manipulation and analysis. This significantly increases the complexity of GIS.

7.1 Summary of Research

The integration of different models attempted in this research is based on a slight modification of the fundamental spatial query. "What is here?" is replaced with "What is closest to here?". The resulting proximal query (a Voronoi diagram) is used to manipulate all main spatial information categories, which simplifies the system and defines relationships between objects. All discrete objects become "fields", with attributes available at all locations. The problem is that in order to represent non-point objects such as networks and polygons, a line segment Voronoi diagram is required or a Constrained Delaunay triangulation is required.

In this research the Line Segment VD has been chosen to represent models with linear features as it appears to more suitable for spatial analysis than the CDT (as discussed in Section 6.1.1). In the CDT the topological and attribute information is all stored together in triangulation edges, while in the LSVD objects are separated from topology. The line segments generate Voronoi regions in the LSVD, just like points generate Voronoi regions in the VD, and the relationships between them are available through Delaunay edges connecting adjacent objects. In the LSVD the line segments and points have proximal region associated with them and these can be grouped to form proximal regions of compound objects. In the CDT only points generate Voronoi cells, because constrained edges are just ordinary triangulation edges with a different "constrained" attribute, and these cells are often not "complete", as some of Voronoi edges might not be available due to applying the visibility property (see Section 3.5). This is why Voronoi regions associated with

endpoints of constrained edges usually do not form proximal regions of objects represented by these edges. Thus the LSVD with objects separated from topology and proximal regions assigned to all objects appears to be a more practical model for spatial analysis than the CDT.

However, implementing the LSVD on digital computers has been shown to be an extremely complex task. Theoretically the LSVD has been studied extensively (Gold (1989); Imai (1996)), but it appears that currently there are only two robust implementations of the LSVD available (Held (2001); Karavelas (2004)). Unfortunately, both these methods allow only construction of the diagram, so the deletion of points and segments is not possible. This prevents dynamic modification of the map and limits the usability of these methods in GIS.

The presence of line segment objects in the LSVD requires modification of the circumcircle calculation method for finding Voronoi vertices to handle linear objects. A modification of the iterative circumcircle method of Anton and Gold (1997) is used in this work, as the direct arithmetic methods (used in Yang and Gold (1995)) failed before due to errors introduced by floating-point inaccuracies, as reported in Gold and Dakowicz (2006). In order to assure the counterclockwise order of the initial circle the input line segments are prepared for the circumcircle calculation by trimming them. Then the circle is calculated iteratively until the requested precision of the result is reached. The resulting circumcircle passes through input points and is tangent to input line segments.

The LSVD construction method proposed in this thesis is based on the moving point idea (Roos (1993); Gold et al. (1997)) where a point is moved throughout the mesh triggering topological events along the trajectory. These are changes in the topology of the diagram observed as flips of Delaunay edges when four objects become cocircular. They are performed to maintain the correctness of the VD/DT at each step of the segment expansion. As the point moves it can also leave behind a line segment as its trace (Gold (1989); Gold et al. (1995)) A new line segment can be created by splitting a new point from an existing one, converting the edge between them to a line segment and expanding this segment by moving its endpoint to its intended location. Linear objects can be removed from the diagram in the same kinetic manner - rolling them up towards their endpoints and flipping edges until the two endpoints are merged into a single point. Since this method allows both the insertion and deletion of point and line objects from the map, local updates are possible at any time. The same algorithm is also used to incorporate linear objects into the DT not as separate objects, but as constrained edges, producing the CDT. It took considerable effort to design and implement these methods and they appear to be robust enough for practical applications.

Depending on the type of the input data and the analysis to be undertaken it is possible to construct layers for GIS using proximal representations of the data provided by the Voronoi diagram, its dual Delaunay triangulation or their line segment based variations - the CDT and the LSVD. Regardless of the type of the input data all these diagrams share the same quad-edge based structure and are created and managed using the same set of operations, so spatial analysis can be performed on these layers in a consistent way.

In the first chapter Figure 1.4 suggested converting all spatial information into Voronoi based models and entering them as layers into GIS. This is expanded in Figure 7.1 presenting the way various input data is added to a GIS based on the LSVD. The four main types of spatial information are converted into various types of Voronoi diagrams and Delaunay triangulations. The resulting diagrams are stored in the GIS as layers and used for selected operations and analyses. Additionally, for all point and linear objects, skeletons and buffer zones can be calculated and used for analysis or as new layers.

The top-left part of Figure 7.1 summarizes the case of discrete objects. The VD created for unconnected points can be used for various proximity analyses, clustering of similar nodes, and other traditional point based operations. Creating a LSVD for discrete polygons adds topology to the model, so the neighbourhood relations are readily available - adjacency of objects is well defined and paths between them can be extracted. Additionally, mobile objects can be incorporated within the model using the moving point mechanism and collisions with objects in the map can be detected.

Networks (the top-right part of Figure 7.1) modelled with the CDT, LSVD or the crust, guarantee connectivity and correctness of the network links and segments. When attributes are added to the segments then such a LSVD can form a basis for any network analyses. Additionally, generating watersheds and calculating catchment areas from river network data using the skeleton method is possible.

When representing polygonal maps (such as choropleths, the bottom-left part of Figure 7.1) as LSVDs, the traditional reclassification and dissolve operations are straightforward. The topology is also available, so it is easy to navigate from one polygon to another. When constructing the map, or changing shapes of polygons, the correctness of the map at the borders of common polygons (avoiding sliver polygons) and the junction nodes (avoiding overshoots and undershoots) can be easily controlled.

The VD/DT based surfaces (the bottom-right of Figure 7.1) can be constructed from various data sources: elevation points, contours or grids. They have many advantages over the traditional grid models and the skeleton method applied for TINs created from contour data produces high quality terrain models without flat triangles. (TIN models can be modified using a cutting plane technique to add dams or simulate road construction, and runoff simulation can can be performed with the method utilising the Voronoi cells as "buckets" storing water.)

Finally, Figure 7.1 illustrates how all these diagrams can be stacked on top of each other and used in a GIS for operations operating on more than one layer. These include "query" (the traditional "point in polygon" operation) and "merge" (the traditional "polygon overlay" operation) operations, that now can be performed in a consistent fashion using any combination of Voronoi diagram based layers. "Query" requires one of the layers to be point based and identifies features in another layer at each point location. When using VDs the operation is straightforward and requires only the Walk routine for each query point. "Merge" combines two layers and consists of two parts - the geometric process and the interpretation of the result. A new layer is loaded into the existing VD, snapping lines or points together when collisions occur and managing intersecting segments. Although not all combinations of input layers are meaningful in GIS, any combination may be created. The result of the overlay can be interpreted in many ways by the user according to his knowledge about the information present in input maps.

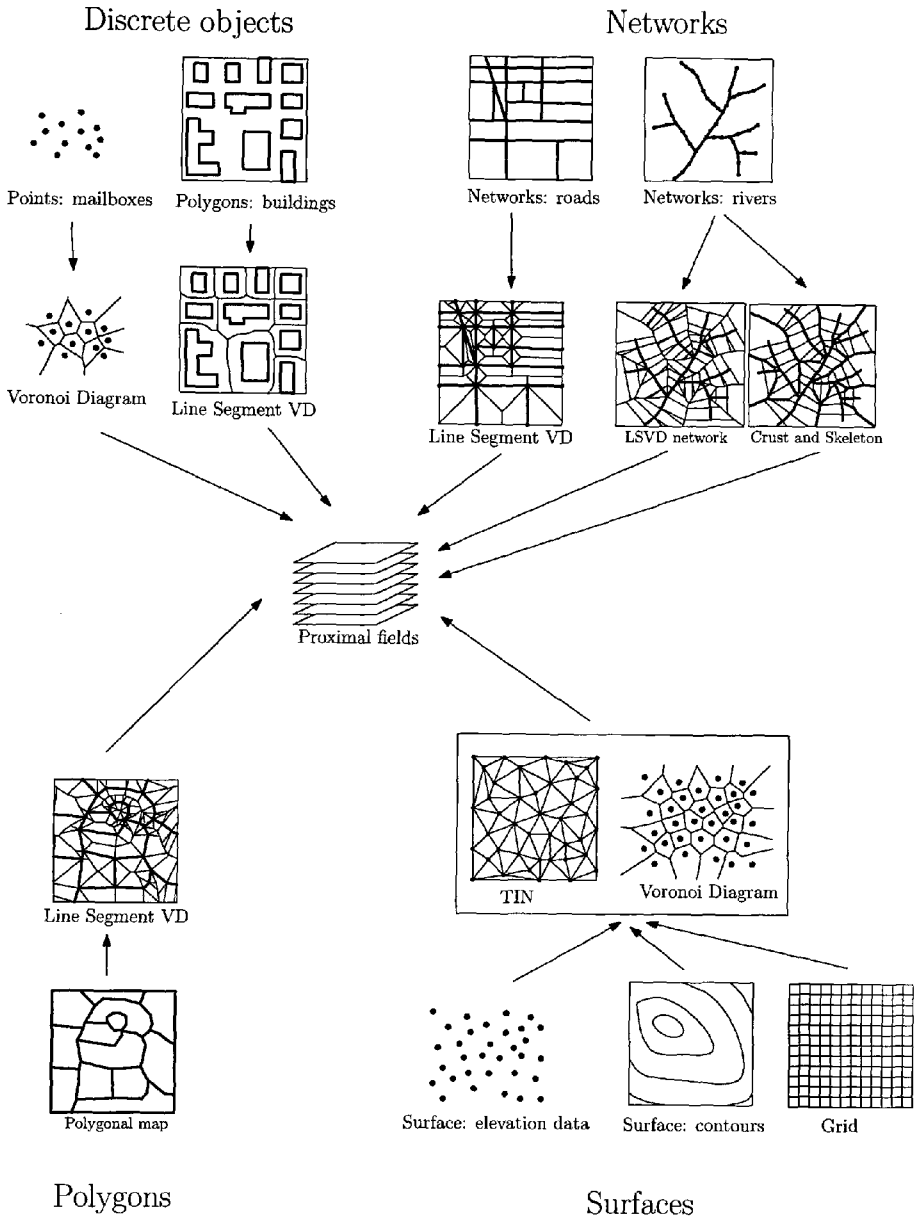


Figure 7.1: The improved GIS model.

7.2 Summary of Contributions

The use of the Voronoi diagram as the data structure for GIS has been advocated by Prof. Gold for many years, but the lack of an operational implementation of a Line Segment Voronoi Diagram, required to representing polygons or networks, prevented turning this idea into reality. The previous attempt of Yang (1997) to construct the LSVD using the kinetic method of Gold (1990) has been unsuccessful. This was due to some problems with calculating Voronoi vertices using direct methods based on calculating intersection points of Voronoi edges. As the result, the Voronoi diagram as a unified structure for GIS has remained a theoretical solution.

In the meantime, Ferrucci et al. (1996) and Anton and Gold (1997) proposed an iterative method for finding a Voronoi vertex for a triple of objects. This method used for point and line

segment input produces a counterclockwise circle located on the correct side of the input segments (which was a problem with the method used by Yang (1997)), centred at the location of the Voronoi vertex.

The LSVD construction method developed in this research combines the kinetic algorithm outlined by Yang (1997) and the iterative Voronoi vertex calculation of Anton and Gold (1997). It allows an interactive construction and modification of the ordinary point based VD, as well as the Line Segment VD. Unlike the methods of Held (2001) and Karavelas (2004) it permits also deletion of points and line segments. It appears to be the first successful implementation of the kinetic method.

Additionally, the same kinetic method is used for the first time to construct and update the Constrained Delaunay Triangulation. Thus, the same set of methods is used to create the ordinary point Voronoi Diagram/Delaunay Triangulation, the Line Segment Voronoi Diagram and the Constrained Triangulation.

While implementing the original iterative Voronoi vertex calculation method of Anton and Gold (1997) for line segments it turned out that occasionally it produced incorrect circles. Thus the initial steps of the algorithm have been modified, so the the input segments are processed first in order to guarantee the correct order of the initial circumcircle, as described in Section 5.3.3. This modification of the algorithm produces a counterclockwise circle for every valid triple of objects.

Another major contribution of this research is actually building a unified spatial model and using it for various GIS analyses. The Voronoi diagram provides a consistent base for all main types of geographic data, as described in Section 6.1. Discrete objects (and networks) become fields with attributes available at all locations and topological relationships established between them. All spatial operations can be performed in the same way, no matter what type of the data is presented in the layer. The overlay operation, one of the fundamental GIS operations, usually executed in GIS different ways depending on the combination of input data, can be performed loading one set data onto another and calculating intersections, regardless the type of the data.The traditional GIS overlay operation is now performed in a consistent manner for all combinations of data input types.

In conclusion, this work proves the viability of using the Voronoi diagram as the basic spatial model for GIS algorithms and analysis. A single spatial model for all main types of spatial data provides a solid algorithmic framework, as well as a clarification of many types of spatial query and analysis that are currently performed with a wide range of frequently inconsistent heuristics. We believe that the proximal Voronoi based model greatly simplifies the formalization of geographic spatial analysis.

7.3 Limitations and Future Work

There are some limitations to this work. The main issue is the robustness of the method, explained in Section 5.7. Although we have found no way to prove that our floating-point based implementation, together with the one or two necessary tolerances, is theoretically robust for all possible geometric configurations, the method is effective in the vast majority of cases and can be used extensively. The method is not optimised for very large data sets and the implementation handles only the amount of data that can fit in the memory of the computer. Also the algorithm is not optimised for speed, as the main concern was to make it operational. However, it is fast enough for most GIS applications. The algorithms have not yet been tested extensively on real world data,

and more testing on various additional data types would be worthwhile, as well as utilising the VD in other GIS operations and applications.

Future work would mainly involve improving the robustness of the method. The main concern is finding a way to deal with approximate positions of Voronoi vertices, which can occasionally create problems when a new point is split from an existing node. Alternatively a new algorithm for computing circumcircles for point and line segment objects not using tolerances and producing precise results could be studied and if possible developed. To monitor the correctness of the diagram at each stage of its construction some kind of sanity tests, like those described by Held (2001), could also be added to the algorithm.

In the current algorithm it is only possible to move one point at a time. An interesting issue would be to manage a simultaneous movement of several points, or construction of multiple line objects at the same time. Investigating and extending the method proposed by Mostafavi (2002) could hopefully add this possibility to the method.

To manage large datasets that do not fit in the memory of the computer some kind of paging mechanism would have to be incorporated into the method. The idea of Voronoi hierarchies described by Gold and Angel (2006), which permits hierarchical indexing, generalization and paging of spatial data, could be further investigated to include line segment objects and used in the method to handle large datasets.

Another extension of the work is to add a possibility of keeping a "log file" of every operation done, as proposed in Gold (1996) and Mioc et al. (1998). This would allow rebuilding the topological state of the diagram from any stage of its modification, using only the original data and the information from the log file.

The kinetic approach could probably be used in 3D to construct the LSVD and the CDT. Extending the 3D moving point method described in Ledoux (2008) with ability to insert line segments or constrained edges would provide an interesting research topic.

Bibliography

Aggarwal A, Guibas LJ, Saxe J, and Shor PW (1989). A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete Comput. Geom.*, 4(6):591–604.

Albers G, Guibas LJ, Mitchell JSB, and Roos T (1998). Voronoi diagrams of moving points. *Internat. J. Comput. Geom. Appl.*, 8:365–380.

Albrecht JH (1996). Universal GIS operations for environmental modeling. In *Third International Conference/Workshop on Integrating GIS and Environmental Modeling*. Santa Fe, New Mexico.

Alliez P, Delage C, Karavelas MI, Pion S, Teillaud M, and Yvinec M (in press). Delaunay Tessellations and Voronoi Diagrams in CGAL. In van de Weijgaert R, Vegter G, Ritzerveld J, and Icke V, editors, *Tessellations in the Sciences: Virtues, Techniques and Applications of Geometric Tilings*. Springer-Verlag.

Alt H and Schwarzkopf O (1995). The voronoi diagram of curved objects. In *SCG '95: Proceedings of the eleventh annual symposium on Computational geometry*, pages 89–97. ACM, New York, NY, USA.

Amenta N, Bern M, and Eppstein D (1997). The Crust and the β-Skeleton: Combinatorial Curve Reconstruction. Research Report, Xerox PARC.

Anglada MV (1997). An improved incremental algorithm for constructing restricted Delaunay Triangulations. *Computers and Graphics*, 21(2):215–223.

Anton F and Gold CM (1997). An iterative algorithm for the determination of Voronoi vertices in polygonal and non polygonal domains. In *Proceedings of the 9th Canadian Conference on Computational Geometry*, pages 257–262. Kingston, Canada.

Augenbaum JM and Peskin CS (1985). On the construction of the Voronoi mesh on a sphere. *J. Computational Physics*, 59:177–192.

Aurenhammer F (1991). Voronoi Diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23:345–405.

Baumgart BG (1975). A polyhedron representation for computer vision. In *Proc. AFIPS Natl. Comput. Conf.*, volume 44, pages 589–596. AFIPS Press, Alrington, Va.

Bern M and Eppstein D (1995). Mesh generation and optimal triangulation. In Du DZ and Hwang FK, editors, *Computing in Euclidean Geometry*, volume 4 of *Lecture Notes Series on Computing*, pages 47–123. World Scientific, Singapore, 2nd edition.

Bernal J (1995). Inserting line segments into triangulations and tetrahedralizations. Technical Report 5596, National Institute of Standards and Technology.

Berry JK (1993). Cartographic modeling: The analytical capabilities of GIS. In Goodchild M, Parks BO, and Steyaert LT, editors, *Environmental Modeling with GIS*, pages 58–75. Oxford University Press.

Blum H (1967). A transformation for extracting new descriptors of shape. In Wathen-Dunn W, editor, *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press.

Boissonnat JD, Faugeras OD, and Le Bras-Mehlman E (1988). Representing stereo data with the Delaunay triangulation. Technical Report 788, INRIA.

Bolstad P (2003). *GIS Fundamentals, A First Text on Geographic Information Systems, Third Edition.* Eider Press, White Bear Lake, Minnesota, USA.

Boots B (1999). Spatial tessellations. In Longley P, Goodchild M, Maguire D, and Rhind D, editors, *Geographical Information Systems: Principles, Techniques, Applications, and Management.*, pages 503–526. John Wiley.

Bossen FJ (1996). Anisotropic mesh generation with particles. Technical Report CMU-CS-96-134.

Botsch M, Steinberg S, Bischoff S, and Kobbelt L (2002). Openmesh – a generic and efficient polygon mesh data structure. In *OpenSG Symposium.*

Bowyer A (1981). Computing Dirichlet tesselations. *Comput. J.*, 24:162–166.

Bruyns CD, Senger S, Menon A, Montgomery K, Wildermuth S, and Boyle R (2002). A survey of interactive mesh-cutting techniques and a new method for implementing generalized mesh cutting using virtual tools. *The journal of visualization and computer animation*, 13:21–42.

Burnikel C, Mehlhorn K, and Schirra S (1994). How to compute the Voronoi Diagram of Line Segments: Theoretical and experimental results. In *ESA '94: Proceedings of the Second Annual European Symposium on Algorithms*, pages 227–239. Springer-Verlag, London, UK.

Burrough P and McDonnell R (1998). *Principles of Geographical Information Systems.* Oxford University Press. New York, NY, USA.

Camara G, Palomo D, Cartaxo R, Souza M, and De Oliveira RF (2005). Towards a generalized map algebra: principles and data types. In *in VII Workshop Brasileiro de Geoinformática. 2005. Campos do Jordão: SBC.*

Chew LP (1986). Building Voronoi diagrams for convex polygons in linear expected time. Technical Report PCS-TR90-147, Dept. Math. Comput. Sci., Dartmouth College, Hanover, NH.

Chew LP (1987). Constrained Delaunay triangulations. In *Proc. 3rd Annu. ACM Sympos. Comput. Geom.*, pages 215–222.

Chin F and Wang CA (1999). Finding the Constrained Delaunay Triangulation and Constrained Voronoi Diagram of a simple polygon in linear time. *SIAM J. Comput.*, 28(2):471–486.

Chrisman N (2002). *Exploring Geographic Information Systems, Second Edition.* John Wiley & Sons.

Couclelis H (1992). People manipulate objects (but cultivate fields): Beyond the raster-vector debate in GIS. In Frank AU, Campari I, and Formentini U, editors, *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, volume 639 of *Lecture Notes in Computer Science*, pages 65–77. Springer-Verlag.

Cova T and Goodchild M (2002). Extending geographical representation to include fields of spatial objects. *International Journal of Geographical Information Science*, pages 509–532.

Crowley W (1985). Free-Lagrange methods for compressible hydrodynamics in two space dimensions. *NASA STI/Recon Technical Report N*, 86.

Dakowicz M and Gold CM (2002a). Extracting meaningful slopes from terrain contours. In Sloot PMA, Tan CJK, Dongarra JJ, and Hoekstra AG, editors, *Proceedings Computational Science - ICCS 2002, Lecture Notes in Computer Science 2231*, pages 144–153. Springer-Verlag Berlin Heidelberg, Amsterdam, The Netherlands.

Dakowicz M and Gold CM (2002b). Visualizing terrain models from contours—plausible ridge, valley and slope information. In *Proceedings International Workshop on Visualization and Animation Of Landscape.* ISPRS Commision V-WG 6, Kunming, China.

Dakowicz M and Gold CM (2003). Extracting meaningful slopes from terrain contours. *International Journal of Computational Geometry and Applications*, 13(4):339–357.

Dakowicz M and Gold CM (2005a). Concepts of interactive terrain modification. In *Proceedings the 13th Annual Conference of GIS Research UK*, pages 252–257. Glasgow, Scotland.

Dakowicz M and Gold CM (2005b). Interactive TIN modification with a cutting tool. In *Proceedings of the 4th ISPRS Workshop on Dynamic and Multi-dimensional GIS (DMGIS05)*, pages 5–9. Pontypridd, Wales, UK.

Dakowicz M and Gold CM (2006). Structuring kinetic maps. In Reidl A, Kainz W, and Elmes G, editors, *Progress in Spatial Data Handling - The 12th International Symposium on Spatial Data Handling*, pages 477–493. Springer-Verlag Berlin.

Dakowicz M and Gold CM (2007a). Finite difference method runoff modelling using Voronoi cells. In *Proceedings of the 5th ISPRS Workshop on Dynamic and Multi-dimensional GIS (DMGIS07)*, pages 55–60. Urumqi, China.

Dakowicz M and Gold CM (2007b). Finite difference runoff modelling using "Voronoi buckets". In *CISIM '07: Proceedings of the 6th International Conference on Computer Information Systems and Industrial Management Applications*, pages 121–124. Elk, Poland.

de Berg M, van Kreveld M, Overmars M, and Schwarzkopf O (1997). *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin.

de By RA, editor (2001). *Principles of Geographic Information Systems.* ITC, Enschede, The Netherlands.

de Smith M, Goodchild M, and Longley P (2008). *Geospatial Analysis - a comprehensive guide. 2nd edition.*

Delaunay B (1934). Sur la sphère vide. A la memoire de Georges Voronoi. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskih i Estestvennyh Nauk*, 7:793–800.

Devillers O (1998). On Deletion in Delaunay Triangulation. Research Report 3451, INRIA.

Devillers O, Pion S, and Teillaud M (2002). Walking in a Triangulation. *International Journal of Foundations of Computer Science*, 13(2):181–199.

Dickerson MT, Drysdale RLS, McElfresh SA, and Welzl E (1997). Fast greedy triangulation algorithms. *Computational Geometry Theory and Applications*, 8(2):67–86.

Dirichlet GL (1850). Über die reduktion der positiven quadratischen formen mit drei unbestimmten ganzen zahlen. *J. Reine Angew. Math.*, 40:209–227.

Edelsbrunner H (2000). Triangulations and meshes in computational geometry. *Acta Numerica*, pages 133–213.

Edelsbrunner H and Tan TS (1992). An upper bound for conforming Delaunay triangulations. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 53–62.

Ervin SM and Hasbrouck HH (2001). *LANDSCAPE MODELING: Digital Techniques for Landscape Visualization.* McGraw-Hill Professional Publishing.

Ferrucci V, Overmars M, Rao A, and Vleugels J (1996). Hunting Voronoi vertices. *Comput. Geom. Theory Appl.*, 6(5):329–354.

Fortune SJ (1987). A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174.

Freundschuh SM and Egenhofer MJ (1997). Human conceptions of spaces: Implications for Geographic Information Systems. *Transactions in GIS*, 2(4):361–375.

Fritts MJ, Crowley WP, and Trease H, editors (1985). *The Free-Lagrange method; Proceedings of the First International Conference, Hilton Head Island, SC, March 4-6, 1985*, volume 238 of *Lecture Notes in Physics, Berlin Springer Verlag.*

Gavrilova M and Rokne J (1999). Swap conditions for dynamic Voronoi diagrams for circles and line segments. *Comput. Aided Geom. Des.*, 16(2):89–106.

Gold CM (1976). Triangular element data structures. In *University of Alberta Computing Services Users' Applications Symposium*, pages 43–54. Edmonton, Canada.

Gold CM (1989). Surface interpolation, spatial adjacency and GIS. In Raper J, editor, *Three Dimensional Applications in Geographic Information Systems*, pages 21–35. Taylor & Francis.

Gold CM (1990). Spatial data structures: the extension from one to two dimensions. In Pau LF, editor, *Mapping and Spatial Modelling in Navigation*, volume 65 of *NATO ASI Series F*, pages 11–39. Springer-Verlag, Berlin.

Gold CM (1991). Problems with handling spatial data—the Voronoi approach. *CISM Journal*, 45(1):65–80.

Gold CM (1994). Advantages of the Voronoi spatial model. In *Proceedings Eurocarto XII*, pages 1–10. Copenhagen, Denmark.

Gold CM (1996). An event-driven approach to spatio-temporal mapping. *Geomatica*, 50(4):415–424.

Gold CM (1999). Crust and Anti-crust: A one step boundary and skeleton extraction algorithm. In *Proceedings 15th Annual Symposium on Computational Geometry*, pages 189–196. ACM Press, Miami, Florida, USA.

Gold CM (2006). What is GIS and what is not? *Transactions in GIS*, 10:505–519(15).

Gold CM and Angel P (2006). Voronoi hierarchies. In *Proceedings of GIScience*, pages 99–111. Munster, Germany.

Gold CM, Charters T, and Ramsden J (1977). Automated contour mapping using triangular element data structures and an interpolant over each triangular domain. In George J, editor, *Proceedings Siggraph '77*, volume 11(2) of *Computer Graphics*, pages 170–175.

Gold CM, Chau M, Dzieszko M, and Goralski R (2005). 3D Geographic Visualization: The Marine GIS. In Fisher P, editor, *Developments in Spatial Data Handling*, pages 17–28. Springer-Verlag, Berlin, Germany.

Gold CM and Dakowicz M (2000). Terrain modelling from contours. In *Proceedings of the 3th ISPRS Workshop on Dynamic and Multi-dimensional GIS*, pages 427–432. Bangkok, Thailand.

Gold CM and Dakowicz M (2005). The crust and skeleton - applications in GIS. In *Proceedings 2nd International Symposium on Voronoi Diagrams in Science and Engineering*, pages 33–42. Seoul, Korea.

Gold CM and Dakowicz M (2006). Kinetic Voronoi/Delaunay drawing tools. In *Proceedings of the 3rd International Symposium on Voronoi Diagrams in Science and Engineering (ISVD06)*, pages 76–84. Banff, Canada.

Gold CM and Dakowicz M (2007). Dynamic cartography using Voronoi/Delaunay methods. In *Proceedings of the 5th ISPRS Workshop on Dynamic and Multi-dimensional GIS (DMGIS07)*, pages 41–47. Urumqi, China.

Gold CM and Maydell U (1978). Triangulation and spatial ordering in computer cartography. In *Proceedings Annual Meeting of the Canadian Cartographic Association*, pages 69–81. Toronto, Canada.

Gold CM, Mioc D, Anton F, Sharma O, and Dakowicz M (2008). A methodology for automated cartographic data input, drawing and editing using kinetic Delaunay/Voronoi diagrams. In Gavrilova M, editor, *Generalized Voronoi Diagram: A Geometry-Based Approach to Computational Intelligence*, pages 159–196. Springer Verlag Berlin Heidelberg.

Gold CM, Nantel J, and Yang W (1996). Outside-in: An alternative approach to forest map digitizing. *International Journal of Geographical Information Science*, 10(3):291–310.

Gold CM, Remmele PR, and Roos T (1995). Voronoi diagrams of line segments made easy. In *Proceedings 7th Canadian Conference on Computational Geometry*, pages 223–228. Quebec City, Canada.

Gold CM, Remmele PR, and Roos T (1997). Voronoi methods in GIS. In van Kreveld M, Nievergelt J, Roos T, and Widmayer P, editors, *Algorithmic Foundations of Geographic Information Systems*, volume 1340 of *Lecture Notes in Computer Science*, pages 21–35. Springer-Verlag.

Gold CM and Snoeyink J (2001). A one-step and skeleton extraction algorithm. *Algorithmica*, 30:144–163.

Goodchild M (1993). The state of gis for environmental problem-solving. In Goodchild M, Parks B, and Steyaert L, editors, *Environmental Modeling with GIS*, pages 8–15. New York: Oxford University Press.

Goodchild MF (1992). Geographical data modeling. *Computers and Geosciences*, 18:401–408.

Goodchild MF (2001). Spatial analysis and GIS. *ESRI User Conference, Pre-Conference Seminar.*

Goodchild MF (2002). Spatial analysis and modeling. In Bossler J, Jensen J, McMaster R, and Rizos C, editors, *Manual of Geospatial Science and Technology*, chapter 3.6, pages 482–499. Taylor and Francis.

Goodchild MF (2005). Geographic information systems. In Kempf-Leonard K, editor, *Encyclopedia of Social Measurement*, pages 107–113. Elsevier Academic Press, Boston, MA.

Goodchild MF, Yuan M, and Cova TJ (2007). Towards a general theory of geographic representation in GIS. *International Journal of Geographical Information Science*, 21(3):239–260.

Goralski IR and Gold CM (2007). Maintaining the spatial relationships of marine vessels using the Kinetic Voronoi Diagram. In *Proceedings of the 4th International Symposium on Voronoi Diagrams in Science and Engineering, ISVD 2007*, pages 84–90. Pontypridd, Wales, UK.

Goralski R, Gold CM, and Dakowicz M (2007). Application of the Kinetic Voronoi Diagram to the real-time navigation of marine vessels. In *CISIM '07: Proceedings of the 6th International Conference on Computer Information Systems and Industrial Management Applications*, pages 129–134. Elk, Poland.

Green PJ and Sibson RR (1978). Computing Dirichlet tessellations in the plane. *Comput. J.*, 21:168–173.

Guibas L and Stolfi J (1985). Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graph.*, 4(2):74–123.

Held M (2001). VRONI: An engineering approach to the reliable and efficient computation of Voronoi Diagrams of points and line segments. *Computational Geometry: Theory and Applications*, 18(2):95–123.

Heller M (1990). Triangulation algorithms for adaptive terrain modeling. In *Proceedings of the 4th International Symposium on Spatial Data Handling*, pages 163–174.

Heywood I, Cornelius S, and Carver S (2002). *An Introduction to Geographical Information Systems. Second Edition.* Pearson Education.

Imai T (1996). A topology oriented algorithm for the Voronoi diagram of polygons. In *Proceedings of the 8th Canadian Conference on Computational Geometry*, pages 107–112. Carleton University Press.

Kallmann M, Bieri H, and Thalmann D (2003). Fully dynamic constrained Delaunay triangulations. *Geometric Modelling for Scientific Visualization.*

Karavelas M (2006). 2D segment Delaunay graphs. In Board CE, editor, *CGAL-3.2 User and Reference Manual.*

Karavelas MI (2004). A robust and efficient implementation for the segment Voronoi diagram. In *International Symposium on Voronoi Diagrams in Science and Engineering (VD2004)*, pages 51–62.

Kemp KK (1993). Environmental modeling with GIS: A strategy for dealing with spatial continuity. Technical report, National Center for Geographic Information and Analysis, University of California at Santa Barbara.

Kjenstad K (2006). On the integration of object-based models and field-based models in GIS. *International Journal of Geographical Information Science*, 20(5):491–509.

Korte GB (2001). *The GIS Book, How to Implement, Manage, and Assess the Value of Geographic Information Systems, 5th ed.* Onword Press, Albany, New York.

Lardin P (1999). *Le diagramme Voronoi generalise comme support a la simulation des ecoulements d'eau souterraine par differences finies integrees. M.Sc. Thesis.* Laval University, Quebec City, Canada.

Lawson CL (1972). Transforming triangulations. *Discrete Math.*, 3:365–372.

Lawson CL (1977). Software for C^1 surface interpolation. In Rice JR, editor, *Math. Software III*, pages 161–194. Academic Press, New York, NY.

Ledoux H (2006). *Modelling Three-dimensional Fields in Geoscience with the Voronoi Diagram and its Dual.* Ph.D. thesis, University of Glamorgan.

Ledoux H (2008). The kinetic 3d voronoi diagram: A tool for simulating environmental processes. In Oosterom PV, Zlatanova S, Penninga F, and Fendel E, editors, *Advances in 3D Geo Information Systems. Lecture Notes in Geoinformation and Cartography. Proceeding of the 2nd International Workshop on 3D Geoinformation, Delft, the Netherlands.*, pages 361–380. Springer-Verlag.

Ledoux H and Gold CM (2006). Voronoi-based map algebra. In Reidl A, Kainz W, and Elmes G, editors, *Progress in Spatial Data Handling - 12th International Symposium on Spatial Data Handling*, pages 117–131. Springer-Verlag.

Lee DT and Lin AK (1986). Generalized Delaunay triangulation for planar graphs. *Discrete Comput. Geom.*, 1:201–217.

Lee DT and Schachter BJ (1980). Two algorithms for constructing a Delaunay triangulation. *Internat. J. Comput. Inform. Sci.*, 9(3):219–242.

Liu Y, Goodchild MF, Guo Q, Tian Y, and Wu L (2008). Towards a general field model and its order in GIS. *International Journal of Geographical Information Science*, 22:623–643(21).

Longley P, Goodchild M, Maguire D, and Rhind D (2001). *Geographic Information Systems and Science.* John Wiley & Sons, 2 edition.

MacNeal R (1953). An asymmetrical finite difference network. *Quarterly of Applied Mathematics*, 11(3):295–310.

Malczewski J (1999). *GIS and Multicriteria Decision Analysis* . John Wiley & Sons.

Mantyla M (1988). *An Introduction to Solid Modeling.* Computer Science Press, Rockville, MD.

Mioc D, Anton F, and Gold CM (1998). Visualizing changes in a dynamic Voronoi data structure via time travel. In *Sixth International Conference in Central Europe on Computer Graphics and Visualization (WSCG'98)*, pages 263–269. Plzen City, Czech Republic.

Mostafavi MA (2002). *Development of a Global Dynamic Data Structure.* Ph.D. thesis, Université Laval, Quebec City.

Mostafavi MA and Gold CM (2004). A global kinetic spatial data structure for a marine simulation. *International Journal of Geographical Information Science*, 18:211–227(17).

Mostafavi MA, Gold CM, and Dakowicz M (2003). Delete and insert operations in Voronoi/Delaunay methods and applications. *Comput. Geosci.*, 29(4):523–530.

Narasimhan TN and Witherspoon PA (1976). An integrated finite difference method for analyzing fluid flow in porous media. *Water Resources Research*, 12:57–64.

Nienhuys H and van der Stappen A (2004). A Delaunay approach to interactive cutting in triangulated surfaces. In Boissonnat J, Burdick J, Goldberg K, and Hutchinson S, editors, *Fifth International Workshop on Algorithmic Foundations of Robotics. Volume 7 of Springer Tracts in Advanced Robotics.*, pages 113–129. Springer-Verlag Berlin Heidelberg, Nice, France.

Okabe A, Boots B, and Sugihara K (1992). *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* John Wiley & Sons, Chichester, UK.

O'Sullivan D and Unwin DJ (2002). *Geographic Information Analysis.* John Wiley & Sons.

Roos T (1993). Voronoi diagrams over dynamic scenes. *Discrete Appl. Math.*, 43(3):243–259.

Sedgewick R (1992). *Algorithms in C++.* Addison-Wesley.

Seidel R (1988). Constrained Delaunay triangulations and Voronoi diagrams with obstacles. Technical Report 260, IIG-TU Graz, Austria.

Shewchuk JR (1996). Robust Adaptive Floating-Point Geometric Predicates. In *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, pages 141–150. Association for Computing Machinery.

Shewchuk JR (1997). Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete and Computational Geometry*, 18:305–363.

Shewchuk JR (1998). A condition guaranteeing the existence of higher-dimensional constrained Delaunay triangulations. In *Symposium on Computational Geometry*, pages 76–85.

Shewchuk JR (2003). Updating and constructing constrained Delaunay and constrained regular triangulations by flips. In *Proc. Symp. Computational Geometry*, pages 181–190. ACM Press.

Sibson R (1980). A vector identity for the Dirichlet tesselation. *Mathematical Proceedings of the Cambridge Philosophical Society*, 87:151–155.

Sibson R (1982). A brief description of natural neighbor interpolation. In Barnett V, editor, *Interpreting Multivariate Data.*, pages 21–36. John Wiley, Chichester, UK.

Sloan S (1993). Fast algorithm for generating constrained Delaunay triangulations. *Computers and Structures*, 47:441–450.

Sugihara K, Iri M, Inagaki H, and Imai T (2000). Topology-oriented immplementation–an approach to robust geometric algorithms. *Algorithmica*, 27:5–20.

Theobald DM (2001). Topology revisited: representing spatial relations. *International Journal of Geographical Information Science*, 15(8):689–705.

Thibault D and Gold CM (2000). Terrain reconstruction from contours by skeleton construction. *GeoInformatica*, 4:349–373.

Thiessen A (1911). Precipitation averages for large areas. *Monthly Weather Review*, 39:1082–1084.

Tomlin CD (1983). A map algebra. In *Proceedings of the 1983 Harvard Computer Graphics Conference*, pages 127–150. Cambridge, MA, USA.

van Oosterom P (1999). Spatial access methods. In Longley PA, Goodchild MF, Maguire DJ, and Rhind DW, editors, *Geographical Information Systems. Volume 1.*, pages 385–400. John Wiley & Sons.

Voronoi GM (1908). Nouvelles applications des paramètres continus à la théorie des formes quadratiques. deuxième Mémoire: Recherches sur les parallélloèdres primitifs. *J. Reine Angew. Math.*, 134:198–287.

Watson D (1981). Computing the n-dimensional Delaunay tesselation with applications to Voronoi polytopes. *Comput. J.*, 24(2):167–172.

Watson DF (1992). *Contouring : a guide to the analysis and display of spatial data.* Pergamon Press, Oxford, UK.

Watson DF and Philip GM (1987). Neighborhood-based interpolation. *Geobyte*, 2(2):12–16.

Winter S (1998). Bridging vector and raster representation in GIS. In *GIS '98: Proceedings of the 6th ACM international symposium on Advances in geographic information systems*, pages 57–62. ACM, New York, NY, USA.

Winter S and Frank AU (2000). Topology in raster and vector representation. *GeoInformatica*, 4(1):35–65.

Worboys M and Duckham M (2004). *GIS: A Computing Perspective, 2nd Edition.* CRC Press, Inc., Boca Raton, FL, USA.

Yang W (1997). *The Design of a Dynamic Spatial Data Management System based on the Voronoi Map Object Model for Sustainable Forestry.* Ph.D. thesis, Laval University, Quebec, Canada.

Yang W and Gold CM (1995). Dynamic spatial object condensation based on the Voronoi diagram. In Chen J, Shi X, and Gao W, editors, *Proceedings of the Fourth International Symposium of LIESMARS'95 - Towards three-dimensional,temporal and dynamic spatial data modelling and analysis,*, pages 134–145.

Yap CK and Dubé T (1995). The exact computation paradigm. In Du D and Hwang FK, editors, *Computing in Euclidean Geometry*, volume 1 of *Lecture Notes Series on Computing*, pages 452–492. World Scientific, Singapore, 2nd edition.

Zubin D (1989). Natural language understanding and reference frames. In Mark D, Frank A, Egenhofer M, Freundschuh S, Mcgranaghan M, and White RM, editors, *Languages of Spatial Relations: Initiative 2 Specialist Meeting Report Technical Paper*, pages 13–16. National Center for Geographic Information and Analysis.

Appendix A

The Circumcircle of Points and Line Segments

This section provides a detailed description of the iterative circumcircle calculation method introduced in Section 5.3.3.

A.1 Introduction to the Method

In the LSVD there are two kinds of objects *OBJ* that can be used to compute the circumcircle - points and segments. When implementing the procedure, both types of objects are stored in the same way, as a pair of points *P1* and *P2* (with *P1.x*, *P1.y* and *P2.x*, *P2.y* coordinates). If *OBJ* is a point, then both *P1* and *P2* refer to that point, so their coordinates are the same (*P1.x* = *P2.x* and *P1.y* = *P2.y*). If *OBJ* is a line segment then *P1* and *P2* store its endpoints so their coordinates are different.

The orientation of segments defined by the order of its endpoints is crucial, as the circle can exist on the left side of the segment. Figure A.1a shows a configuration of two points *Pa* and *Pb* and a line segment *LSc* with *Pc1* and *Pc2* endpoints. Both points are located on the left side of the segment and there is a circumcircle *Pa*, *Pb*, *LSc* on the left side of *LSc* tangent to the segment and passing through *Pa* and *Pb* points. Figure A.1b shows a similar configuration and the only difference is the opposite orientation of the segment. In this case both points *Pa* and *Pb* are located on the right side of the segment and it is not possible to find a counterclockwise circle *Pa*, *Pb*, *LSa* for such a configuration.

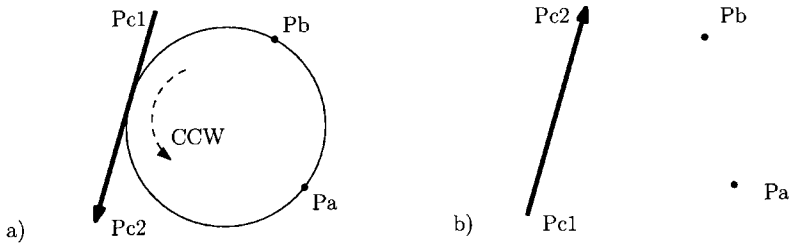


Figure A.1: Two configurations of a line segment and two points. a) Both point on the left side of the segment.b) Both points on the right side of the segment.

The algorithm consists of two main parts. The first is trimming of the input segments to

guarantee a counterclockwise (cww) order of the initial feet and the second is the iterative process of calculating circles. Additionally the order of the input objects is set according to the x, y coordinates and cases of objects being line segments and their endpoints are handled separately. All those operations are described in detail below.

A.2 The Initial Steps

The first part of the procedure consists of reordering the input objects to guarantee the same result of the process. Additionally, the case of the line segment and its endpoint is handled by calculating the circle using direct geometric operations, not by the iterative method.

Due to the floating point arithmetic used in the project, geometric calculations on a differently ordered set of points can produce different results. To avoid that the objects have to be passed to the iterative part of the procedure in a coherent order, for example according to their coordinates. The object having the smallest x and y coordinates values is set as the first one. In cases of line segments their middle points are used for the comparison. For example reordering objects *g, h, j* may lead to *h, j, g* (if *h* is the first object according to x and y coordinates) or *j, g, h* (if *j* is the first) sequences.

In the case of three counterclockwise objects being a line segment *OBJa = (Pa1, Pa2) = LSa* (with *Pa1 <> Pa2*), its endpoint *OBJb = (Pb1, Pb2) = Pb* (with *Pb1 = Pb2* and *LSa.Pa2 = Pb*) and any third object *OBJc* the iterative process is not used. Instead, the circle is calculated using direct geometric operations, as it centre lies on a line perpendicular to the line segment passing through its endpoint. The result is a circle tangent to the line at the endpoint position passing through or touching the third object, as in Figure A.2 where a point is the third object. The circumcentre *CC* projects on the line segment exactly at its endpoint position (although due to floating point arithmetic it is not always the case). This solution for such configurations of the input objects is simple and much faster and precise than the iterative process, which would require many iterations to calculate the circle.

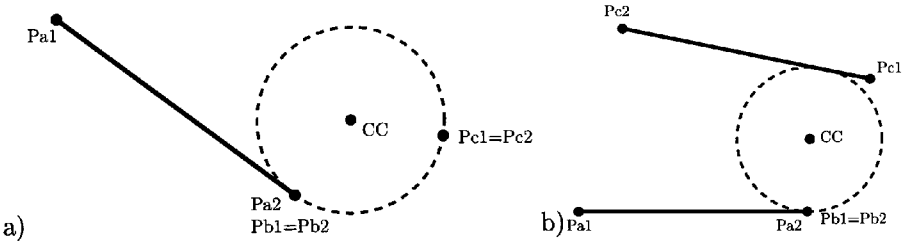


Figure A.2: Circumcircles of line segments with their endpoints and another object. a) The third object is a point. b) The third object is a segment.

A.3 Processing Objects

After performing the initial tests, the actual iterative circumcircle process can be started. The initial circle is going to be computed using the input points or the middle points of line segments. However, using these middle points directly (even if they are counterclockwise oriented) may lead to a clockwise orientation of the initial points so a ccw circumcircle would not exist. To avoid such situations a removal of parts of the segments is often necessary. This is depicted in Figure A.3

showing three segments and their initial clockwise oriented middle points *a1*, *b1* and *c1*. After trimming two intersecting line segments their new middle points *b1t* and *c1t* are computed. The orientation of *a1*, *b1t* and *c1t* is counterclockwise and those points can be used to compute the initial circle.

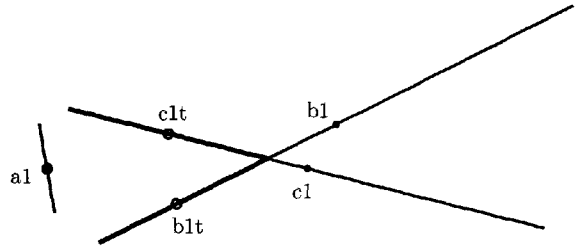


Figure A.3: Middle points of the original and trimmed line segments.

To avoid clockwise ordered starting points for the initial circle, if any of the input objects is a line segment, the line segment objects need to be prepared for the iterative processing by trimming them. There are four possible combinations of the three input objects:

1. Three points (*OBJa, OBJb, OBJc = POINT*), as in Figure A.4a.
2. Two points and one line segments (*OBJa, OBJb = POINT; OBJc = SEGMENT*), as in Figure A.4b.
3. One point and two line segments (*OBJa = POINT; OBJb, OBJc = SEGMENT*), as in Figure A.4c.
4. Three line segments (*OBJa, OBJb, OBJc = SEGMENT*), as in Figure A.4d.

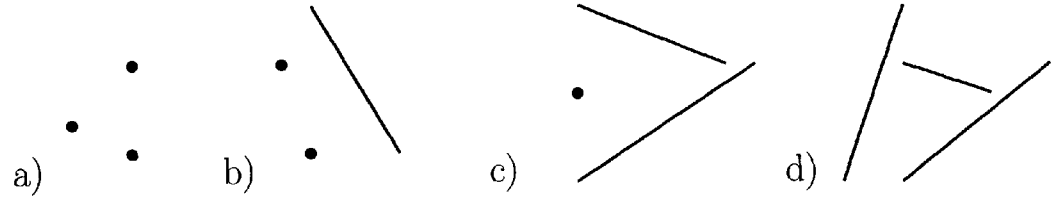


Figure A.4: Four different configurations of points and line segment for the circumcircle calculation process.

Each of the combinations is detected as in the code below and processed separately.

```
if OBJa=SEGMENT then
  if OBJb=SEGMENT then
    if OBJc=SEGMENT then
      ProcessThreeSegments(OBJa,OBJb,OBJc)
    else
      ProcessOnePointTwoSegments(OBJc,OBJa,OBJb)
  else
    if OBJ3=SEGMENT then
      ProcessOnePointTwoSegments(OBJb,OBJc,OBJa)
    else
      ProcessTwoPointsOneSegments(OBJb,OBJc,OBJa)
else
```

```
if OBJ2=SEGMENT then
  if OBJ3=SEGMENT then
    ProcessOnePointTwoSegments(OBJa,OBJb,OBJc)
  else
    ProcessTwoPointsOneSegment(OBJc,OBJa,OBJb)
else
  if OBJ3=SEGMENT then
    ProcessTwoPointsOneSegment(OBJa,OBJb,OBJc)
  else
    ProcessThreePoints(OBJa,OBJb,OBJc);
```

Processing the first case of three points does not require any operations and the circumcircle is computed using the ordinary method. The three other cases will be described in detail. Additionally, processing of a pair of segments will be described separately, as it is an operation used in each of the three cases with line segments.

A.3.1 Processing Two Segments

A line divides the plane into two half-planes. For two oriented, infinite and non-parallel lines there can be defined four areas of the plane sharing one common point - the intersection point of the lines. If we want to define a circle tangent to two lines in counterclockwise order it would be located in the area being the common left area of two lines, no other of those four areas makes such a circle possible.

For two oriented segments (that can be extended to lines) a counterclockwise circle can be defined if there are at least parts of both segments located along the parts of the infinite lines bordering the common left area of the lines. The segments should intersect or be located on the left sides of each other.

The idea of the method is to test if the configuration of the segments makes computing the counterclockwise circles touching them possible and if so to extract parts of the segments where such circles can exist by trimming their unwanted parts.

Two segments *OBJa = (OBJa.P1, OBJa.P2) = (Pa1, Pa2)* and *OBJb = (OBJb.P1, OBJb.P2) = (Pb1, Pb2)* are processed. Segments intersect if their endpoints are on the opposite sides of each other. This can be tested using four determinants, as two determinants are needed for each segment to test if the endpoints of the other segment are on its opposite sides.

The ProcessTwoSegments method starts with computing four determinants *d1, d2, d3* and *d4*. Firstly, for the segment *OBJa* the location of both endpoints *Pb1* and *Pb2* of *OBJb* against *OBJa* is determined (*d1* and *d2*). If the signs of *d1* and *d2* are different then *Pb1* and *Pb2* are on the opposite sides of *OBJa*. If both *d1* and *d2* are positive then the whole segment *OBJb* is on the left side of *OBJa*, otherwise, if both *d1* and *d2* are negative then the whole segment *OBJb* is on the right side of *OBJa*. The endpoints of *OBJa* are tested against the segment *OBJb* in the same way using *d3* and *d4* determinants.

```
d1=Det(Pa1,Pa2,Pb1);
d2=Det(Pa1,Pa2,Pb2);
d3=Det(Pb1,Pb2,Pa1);
d4=Det(Pb1,Pb2,Pa2);
```

If any of the pairs *d1- d2* or *d3 - d4* is negative then no circle is possible, as the whole segment is on the right side of another one.

If all determinants are zero then those are two collinear segments and no further processing is needed. If *d1* and *d4* are zero then segments share a common point *Pa2* = *Pb1* and no further processing is needed. The same in case of *d2* and *d3* equal zero, when *Pa1* = *Pb2*.

Otherwise the segments intersect or one or both are positioned completely on the left side of another segment. The intersection point *IP* (with *IPx, IPy* coordinates) of lines defining those segments is calculated. Then trimming of the segments is attempted to remove unwanted parts of segments on the right side of the intersection point.

```
if (d1>0) and (d2<0) then
  Pb2 := IP
else
if (d1<0) and (d2>0) then
  Pb1 := IP;
if (d3>0) and (d4<0) then
  Pa2 := IP
else
if (d3<0) and (d4>0) then
  Pa1 := IP;
```

If *Pb1* endpoint is on the left side of *OBJa* segment *(d1>0)*, and *Pb2* endpoint is on the right side *(d2<0)*, then the right side of *OBJb* is trimmed, by changing coordinates of *Pb2* to the coordinates of the intersection point *IP*. The resulting *OBJb* has *Pb1* and *Pb2* = *IP* endpoints. In the same configuration of segments, if the orientation of *OBJb* is the opposite (*d1<0* and *d2>0*), then its *Pb1* endpoint is trimmed to *IP*. The analogical tests and operations are performed on the segment *OBJa*. As a result the segments have no parts on the right sides of each other.

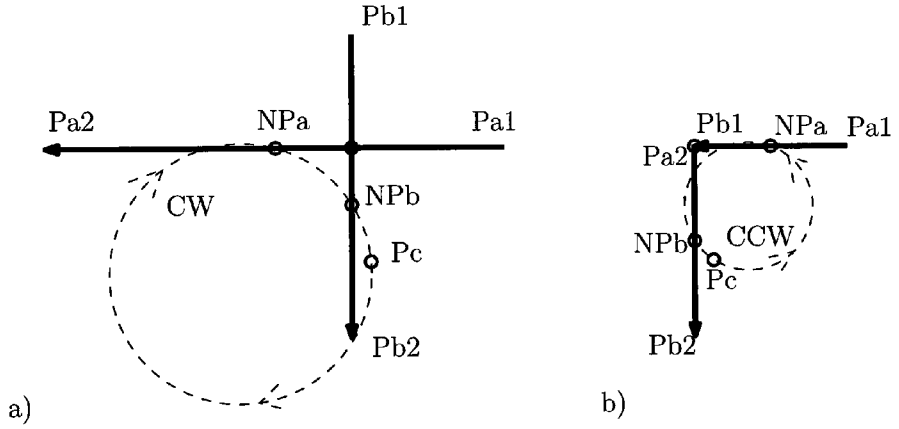


Figure A.5: The result of trimming endpoints of two line segments.

Figure A.5a shows a configuration of two segments *OBJa* and *OBJb* and a point *OBJc* = *Pc*. The middle points of two segments *NPa* and *NPb*, together with the point *Pc* are used as the defining points for the initial circumcircle. Without trimming these segments the orientation of the circle *OBJa, OBJb* and *OBJc* is clockwise, which leads to the wrong result. Figure A.5b shows the same segments after trimming them at the intersection point. The circumcircle defined by two new middle points and *Pc* is counterclockwise now.

A.3.2 Processing Two Points and One Segment

Two points *OBJa* and *OBJb* are processed, together with a segment *OBJc*. Firstly, the cases when one of the points is the endpoint of the segments are processed. If *OBJa* or *OBJb* is a wrong endpoint of *OBJc* (*OBJa* = *OBJc.P1* or *OBJb* = *OBJc.P2*) there is no circle possible as the order of objects is always clockwise in such case. If *OBJa* or *OBJb* is a correct endpoint of the segment *OBJc* (*OBJa* = *OBJc.P2* or *OBJb* = *OBJc.P1*) then it is tested if the other point is on the left side of the segment. If so the other endpoint of the segment is trimmed to the point where that point projects onto it, otherwise no circle is possible for those objects.

If none of the points is an endpoint of the segment then it is tested if both points are on the left side of the segment. If so a temporary segment *OBJt* is created from those two points (*OBJt.P1* = *OBJa* and *OBJt.P2* = *OBJb*) and both segments are processed. This leads to trimming the segment *OBJc* with *OBJt* if possible or rejecting the circle.

A.3.3 Processing One Point and Two Segments

One point *OBJa* = *Pa* and two segments *OBJb* and *OBJc* are processed. A circle is not possible if the point lies on the right side of any of the segments or *Pa* is a wrong endpoint of one of the segments (*OBJb.P2* = *OBJa* or *OBJc.P1* = *OBJa*). Firstly those two segments are processed, which identifies incorrect configurations or leads to trimming them. Then if *Pa* is a correct endpoint of one of the segment no further processing is required. If not, a trimming of segments with *Pa* is attempted. If they are parallel then their endpoints are trimmed to the points where *Pa* projects onto them, otherwise the intersection point *IP* is computed and *Pa* is projected onto both segments. If any of the projection points *PRb* and *PRc* falls inside the segment then trimming is performed. Distances from *PRb* and *PRc* to *IP* are calculated, the larger value *d* is selected and points are located on both segments in the distance *d* from *IP*. If any of these points falls inside the segment then that segment is trimmed to that point.

Figure A.6 shows a configuration of one point *Pa* and two segments *OBJb* = *(Pb1, Pb2)* and *OBJc* = *(Pc1, Pc2)*. The segments intersect at the *IP* point. Their endpoints *Pb2* and *Pc1* are trimmed to that point and *Pbm* and *Pcm* are the middle points of the resulting segments. The orientation of *Pa*, *Pbm* and *Pcm* is clockwise (circle *C1*) so those points cannot be used for calculating the circumcircle. The point *Pa* is projected onto both segments, which produces *PRb* and *PRc* points. The segment *OBJb* cannot be trimmed at *PRb* point, as the resulting circle would not be tangent to the segment. That is why a new point *PRbc* is calculated on *OBJb* segment in the the same distance from *IP* as *PRc* point (which is further away from *IP* than *PRb*). Both segments are trimmed, so *OBJb* = *(PRbc, IP)* and *OBJc* = *(IP, PRc)*. The new middle points *Pbm1* and *Pcm1* are calculated for both segments. The orientation of *Pa, Pbm1* and *Pcm1* is counterclockwise, so a new circumcircle *C2* = *(Pa, Pbm1, Pcm1)* can be calculated.

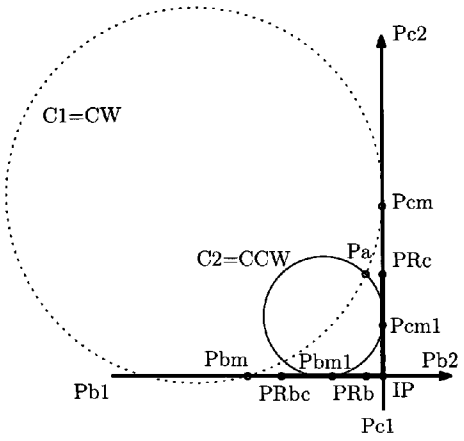


Figure A.6: Trimming two segments with a point.

A.3.4 Processing Three Segments

Processing three segments *OBJa*, *OBJb* and *OBJc* is performed by processing each pair of them separately. If any of the pairs is not valid then the circle is not possible for this configuration of the segments. Otherwise segments are trimmed with their intersection points if possible.

```
if  ProcessTwoSegments(OBJa,OBJb)
and ProcessTwoSegments(OBJb,OBJc)
and ProcessTwoSegments(OBJc,OBJa) then
  result := true
else
  result := false;
```

A.4 Iterative Circumcircle Computing

In the next step the initial circle *C1* is computed for the three processed objects. The points or midpoints of line segments are used as the input for the circumcircle procedure.

```
if OBJx is a Point then
  C1x := OBJx
else
if OBJx is a Line Segment then
  C1x := (OBJx.P1 + OBJx.P2)/2;
```

If the order of the input points is counterclockwise the resulting circle *C1* has the circumcentre *CC1* at the *cx,cy* location and its radius is *CR1*, otherwise there is no circle for such a configuration of objects and the process cannot be continued.

```
Circumcircle(C1a,C1b,C1c, CC1,CR1);
```

Then the initial circle *C1* is used to obtain a new set of feet for the line segment objects (point objects' feet do not change during the whole process). The circumcentre *CC1* is projected onto each of the line segments. If the projection point *CC1xp* falls inside the segment x then a point half way between the previously used foot *C1x* and *CC1xp* is used as a new foot *C2x* for the next circumcircle. Otherwise, if the projection falls outside the segment then the new foot is selected between the endpoint of the segment closest to *CC1xp* and the previously used foot *C1x*.

```
if OBJx is a Point then
  C(i+1)x := OBJx
else
if OBJx is a Line Segment then
begin
  t := ProjectionResult(CC(i), OBJx);
  if t<=0 then
    C(i+1)x := (C(i)x + OBJx.P1)/2
  else
  if t>=1 then
    C(i+1)x := (C(i)x + OBJ.P2)/2
  else
  begin
    CC(i)xp := point on OBJx at t location
    C(i+1)x := (C(i)x + CC(i)xp)/2;
  end;
end;
```

The middle points are used as new feet, because using the projection points directly for the new circle can lead to false results. This can be seen in Figure A.7, where the circumcentre *CC1* of the initial circle *C1* is projected onto each of the three segments *LSa*, *LSb* and *LSc*. This produces three new points *CC1ap*, *CC1bp* and *CC1cp* to be used to calculate the next circle. However their orientation is clockwise, since the projection point *CC1bp* lies outside the segment *LSb*, so the circle cannot be computed. One solution could be to use the endpoint of the segment when the projection point falls behind it. However, if both of them fall outside joined line segments (imagine segments *LSb* and *LSc* in Figure A.7 joined at a sharper angle) this would lead to two feet at the same location, so it would not be possible to compute the circle. The solution is to use a point located between the previous foot and the circumcircle projection. This guarantees a correct orientation of the new circle.

In the next step the new circumcircle is calculated, with a *NCC* circumcentre and *NR* radius, using the new set of feet. The distance between the previous circumcentre *OCC* and the new *NCC* is computed. If it is shorter than a specified value *VAL* the process terminates. Otherwise a new set of feet is obtained, the next circumcircle is computed and compared with the last one until two circumcentres are close enough to terminate the process.

```
Circumcircle(C(i)a,C(i)b,C(i)c, CC(i),CR(i));
while true do
begin
  set C(i+1)a on OBJa using CC(i);
  set C(i+1)b on OBJb using CC(i);
  set C(i+1)c on OBJc using CC(i);
  Circumcircle(C(i+1)a,C(i+1)b,C(i+1)c, CC(i+1),CR(i+1));
  d := distance(CC(i+1),CC(i));
  if d<VAL then
    result := circle(CC(i+1),CR(i+1))
  else
    inc(i);
end;
```

Figure A.7 shows the process of finding the circumcircle of three connected line segments *LSa = (Pa1,Pa2)*, *LSb = (Pb1,Pb2)* and *LSc = (Pc1,Pc2)*.

Firstly three middle points *C1a*, *C1b* and *C1c* of each of the segments are selected as feet and used to calculate the initial circle *C1*. In the next step the circumcentre *CC1* of *C1* is projected

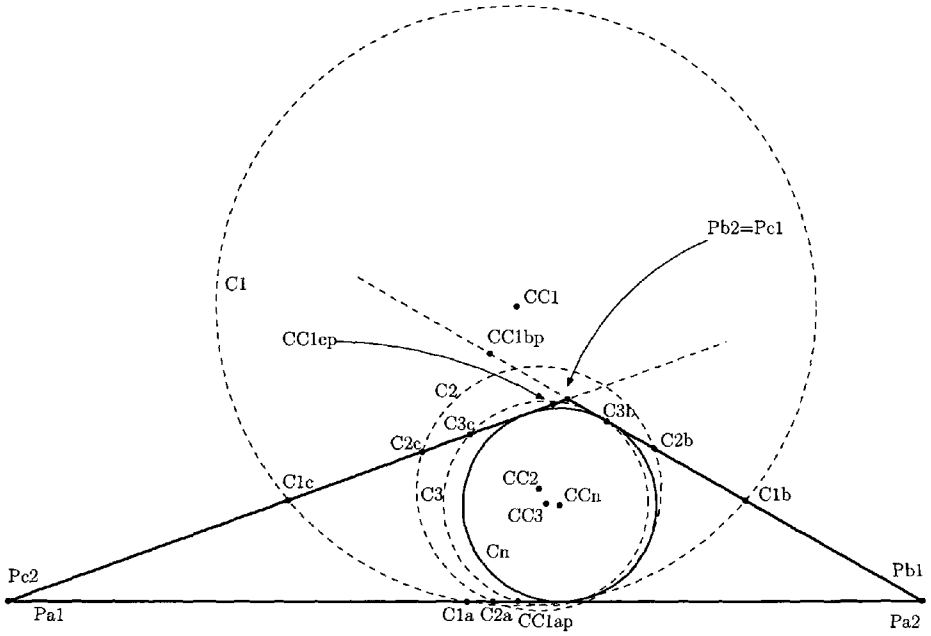


Figure A.7: Iterations of the circumcircle computing process.

onto each segment. Projecting *CC1* onto *LSa* results in a point *CC1ap*, which lies on the segment. Points *C1a* and *CC1ap* are used to compute the next point *C2a* in the middle of them. The second projection point *CC1bp* of *CC1* falls outside *LSb* (behind its *Pb2* endpoint) so the new point *C2b* is computed between *Pb2* and *CC1b* points. The third projection point *CC1cp* falls inside *LSc* segment so *C2c* is calculated between *C1cp* and *C1c*. The three new points are used to calculate the second circumcircle *C2* with the circumcentre at the *CC2* location. Using these feet directly would have led to a clockwise order of points. Then a distance between *CC1* and *CC2* is calculated and compared with a pre-specified tolerance value. Since it is not small enough the circle *C2* is set as the current circle and the new feet for the next circle *C3* are computed projecting *CC2* onto each of the segments. This time all projection points fall inside each segment, so middle points between them and previous feet are used. The process ends after n iterations, when two consecutive circumcentres are closer than the specified tolerance value. The resulting circle *Cn* with the circumcentre *Cn* tangent to each of the segments is marked with a solid line.

This method can be also used to find circles tangent to lines passing through line segments, as shown in Figure A.8. The idea is to extend the input line segments using the circumcircle iteratively calculated for them and then use these extended segments to calculate iteratively the final circle. Using the configurations of segments in Figure A.8, such a circle can be calculated by projecting the circumcentre *CC1* of the circle *C1* (calculated for the initial line segments) onto the extensions of two segments *Pb1*, *Pb2* and *Pc1*, *Pc2* and extending them to those projection points *Pa1C1* and *Pc2C1*. Then those extended segments can be used to calculate iteratively the new circle *C2*. This circle is guaranteed to be tangent to all three segments, as its radius is smaller than the radius of the initial circle *C1*.

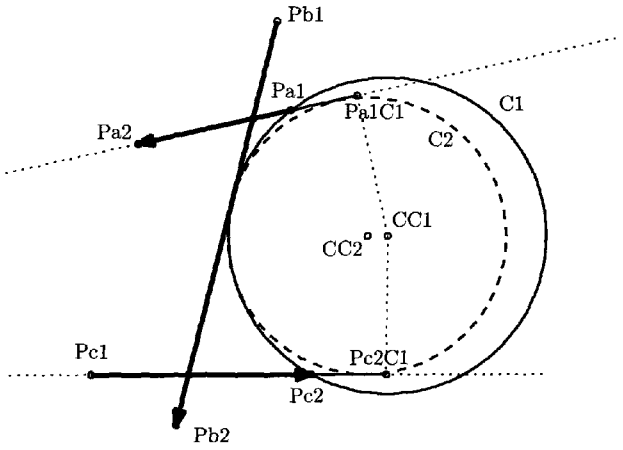


Figure A.8: The circumcircle not tangent to the input segments.

Appendix B

Moving Points in DT/CDT/LSVD

This section provides a detailed description of the MovePoint method introduced in Section 5.5.

B.1 Introduction to the Method

In the moving point operation "MovePoint" a point *MP* is moved from the origin *OP* to the destination *DP* location. Primarily the method has been developed in order to allow relocation of existing nodes of the mesh (Gold (1990)), but later this idea was found useful to insert line segments into triangulations (Gold et al. (1997)). In this work MovePoint is used to relocate existing nodes and to expand and retract line segments and constrained edges. Relocation of existing nodes is a part of point insertion and deletion operations. The insertion consists of two operations: splitting a new node from an existing one and relocating it to the desired location. Similarly, the deletion consist of relocating the deleted node towards a selected node and merging them together.

Summarizing, the point movement operation is used to:

1. Move a point from *OP* to *DP* location. The moving point *MP* can be the origin point *OP* (this relocates *OP*) or a new point split from *OP* (this adds a new point).

2. Expand a constrained edge *CE* from *OP* to *DP*. *MP* is split from *OP*, the edge between them is marked as constrained (so *MP* becomes one of the endpoints of *CE*) and *MP* is moved to the location *DP*, expanding *CE*.

3. Expand a line segment *LS* from *OP* to *DP*. *MP* is split from *OP*, the edge between them is converted to a line segment *LS* (so *MP* becomes one of the endpoints of *LS*) and *MP* is moved to the location *DP*, expanding *LS*.

4. Retract a constrained edge *CE* from *OP* to *DP*. The moving endpoint *MP* can be split from *OP* (so *OP* remains in the mesh after the retraction) or *OP* can be *MP* (if *CE* is not connected to any other constrained edge this removes *CE* together with the endpoint *OP*). *MP* is moved to the location *DP*, retracting the constrained edge.

5. Retract a line segment *LS* from *OP* to *DP*. *MP* can be *OP* (so *LS* is retracted together with its endpoint *OP*) or a new endpoint of *LS* split from *OP*, as in the case of retracting constrained edges.

All those operations are performed using one procedure MovePoint, which consists of three main parts. In the first part the initial tests (including collision detection at the origin, destination and along line segments and constrained edges connected to *MP*) and operations (including Split if needed) are performed. Then the topological events are processed by testing intersections with the trajectory of the real and imaginary triangles adjacent to *MP*. In the third part *MP* is relocated to the location of the nearest topological event and an appropriate edge is swapped. Those two parts are repeated until there are no topological events detected along the trajectory and then *MP* is relocated to the final destination location.

Although MovePoint procedure is used for all possible moving point scenarios, some operations and tests are only performed for particular scenarios. The operations are distinguished by the byte "*operation*" argument and depending on its value MovePoint performs different operations and produces different results:

- *operation* = 1 - move *OP* to *DP*, so *MP*=*OP*,
- *operation* = 2 - split *MP* from *OP* and move to *DP*,
- *operation* = 3 - make a constrained edge from *OP* to *DP*, *MP* is split from *OP*,
- *operation* = 4 - make a line segment from *OP* to *DP*, *MP* is split from *OP*,
- *operation* = 5 - retract a constrained edge or line segment from *OP* to *DP* together with *OP*, so *MP*=*OP*,
- *operation* = 6 - retract a constrained edge or line segment from *OP* to *DP* leaving *OP*, so *MP* is split from *OP*.

Retraction of line segments and constrained edges requires one more parameter "*retractedObject*" in MovePoint procedure. The parameter is a quad-edge that identifies what kind of feature is being removed. When removing a constrained edges *retractedObject* parameter is that edge directly, and when a line segment *LS* is being retracted then the destination of *retractedObject* (*retractedObject.Dest*) points to *LS*. This parameter is also used in Split (when retracting a constrained edge) and DisconnectLineSegment (when retracting a line segment) in the operation number 6. In operations 1-4 the parameter *retractedObject* is set to nil.

The last of the parameters is a Boolean flag "*reconstruction*" , which is set to true in operations 3 and 4 when line segments or constrained edges are reconstructed after splitting them due to collisions. In all other cases this parameter is set to nil. When this flag is set to true the collision tests are not performed during the process, as the feature has already existed in that location before it was retracted.

Summarizing the above information the procedure call is MovePoint(*Origin Point, Destination Point, operation, retractedObject, reconstruction*).

The point *MP* can be a single point (when relocating a single node) or an endpoint of an existing constrained edge or line segment (when expanding or retracting them). The trajectory is a straight line defined by the *OP* and *DP* locations. However, the final trajectory can be modified if *MP* collides with other objects. If *MP* collides with a point *CLP* during the movement, then *MP* is moved back to *OP* and then moved towards *CLP*, and then from *CLP* towards *DP*, so the resulting trajectory is *OP-CLP-DP*. Also if *MP* collides with a line segment or a constrained edge (so they intersect the trajectory), then *MP* is relocated back to *OP*, the colliding edge or segment

is split into two parts with a new point *CLP* at the collision location. Then *MP* is moved towards *CLP* and from *CLP* to *DP*.

Figure B.1 shows an example situation when the trajectory is modified. A line segment is attempted to be inserted between two existing points *OP* and *DP* (Figure B.1a). During the expansion of the line a collision is detected with *CLP* point. The line is shrunk back to *OP* and a new line is constructed between *MP* and *CLP* and then between *CLP* and *DP* (Figure B.1b).

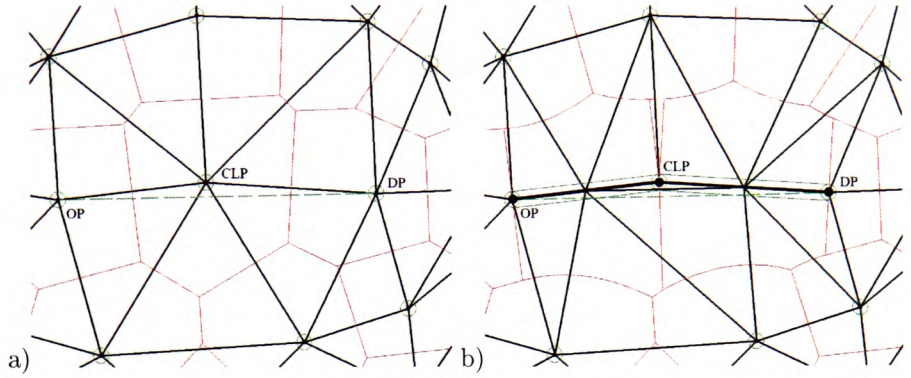


Figure B.1: The trajectory modification due to a collision with a point. a) The initial trajectory between OP and DP. b) The resulting line segment created after the collision with CLP.

The method consists of three main parts. In the first part the initial tests and operations are performed. This is performed only once, at the beginning of the movement. Then the topological events are processed by testing intersections of the real and imaginary triangles with the trajectory. After that the point is relocated to the location of the nearest topological event and an appropriate edge is swapped. Those two parts are repeated until there are no topological events detected along the trajectory. Finally the point is relocated to the destination.

The description below concerns the point movement and constrained edge and line segment expansion (operations 1-4). The retraction operations 4-5 were covered in Section 5.5.2.

B.2 The Initial Tests and Operations

Only single points can be moved without splitting a new point, so it is tested if *MP* is not an endpoint of a line segment or a constrained edge.

If the coordinates of *OP* and *DP* are equal or the distance between *OP* and *DP* is smaller than *2*DiskRadius* then no movement is performed, as the destination and the current location are the same.

Then collision tests are performed at *DP* location. If there is a point *P* within the range of *2*DiskRadius* then *DP* coordinates are changed to that point coordinates. In the case of splitting *MP* from a line segment's endpoint it is also tested if *P* is not the other endpoint of that line segment and in such a case the procedure terminates here with *P* as the result.

If there is a line segment *LS* colliding with the *DP* location then a variable *CollWithHL* is set to true and *LS* is split into two segments at the location of the projection of *DP* onto *LS*. If *LS* had *OP* as its endpoint the procedure terminates here, otherwise *DP*'s coordinates are changed to the point where *LS* was split.

If there is no collision with a point or a line segment then it is tested if *DP* collides with constrained edges. The edges originating at the vertices of the triangle enclosing the *DP* location

are tested to see if any of them does not collide with *DP*. The distance d between *DP* and each constrained edge is calculated and if it is smaller than *2*DiskRadius* the edge collides with *DP*. The edge *e* for which the *d* value is the smallest is selected as the one that collides with *DP* directly and is split into two edges at the location of the *DP* projection. The new point on that edge becomes the new destination *DP*.

Then it is tested if there are no line segments connected to *OP* having the other endpoint colliding with the trajectory, to use one of them as the new starting point for the point movement or the line segment expansion. This eliminates the possibility of moving points along or very near existing line segments. For each line segment having *OP* as one endpoint the distance from the other endpoint *P* to the trajectory is calculated and if it is smaller than *2*Disk Radius* then *P* collides with the trajectory. However, it is possible that there are multiple endpoints colliding with the trajectory, as in Figure B.2, where *P1* and *P2* endpoints collide with the *OP-DP* trajectory. Additionally not every colliding endpoint is visible from the trajectory, some of them might be obscured by other line segments, like point *P2* in Figure B.2. In order to detect if a point *P* collides directly with the trajectory it is tested if there are no other line segments between *P* and the trajectory. Firstly the orientation of *OP*, *P* and *DP* is computed. If CCW=0 then the trajectory overlaps that line segment and *P* can be used immediately as the new starting point for the movement. If CCW<0 then *P* lies on the left side of the trajectory and in order to use *P* as the new starting point there cannot be any lines on its right side, between *P* and the trajectory. So neighbours on the right side of *P* are visited using *Oprev* until the first object giving the answer is found. Points and endpoints of line segments are tested using the orientation test again (CCW*(OP, test point, DP)*) and if there is a point or an endpoint found on the other side of the trajectory with CCW>0 then *P* directly collides with the trajectory and can be used as a potential new starting point for the movement. Alternatively, there can be a situation, when the first neighbouring object is a line segment with the endpoint on the same side of the trajectory as *P*. Such a situation is depicted in Figure B.2 for the endpoint *P2*, which is detected to collide with the trajectory. CCW*(OP, P2, OP)*<0 so *P2* is on the left side of *OP-DP*. Neighbours on the right side are visited using the operator *Oprev* and *P1* endpoint is tested. CCW*(OP, P1,DP)*<0 so *P1* is also on the left side of the trajectory, colliding with it as well and separating *P2* from it, so *P2* cannot be used as the new starting point. The analogous situation is when CCW>0. In such a case *P* lies on the right side of the trajectory and neighbours of *OP* on the left side are tested in the same way as before only using the operator *Onext* and the orientation test.

After detecting a collision of the trajectory with one or more of endpoints of adjoining segments, the endpoint *P* closest to the trajectory is selected as the new starting point for the movement (an alternative option is to select the first endpoint colliding with the trajectory). The moving process starts again, this time from *P* towards *DP*. This modifies the trajectory from *OP-DP* to *P-DP*. Analogous tests are performed for constrained edges.

If we analyse Figure B.2 the initial trajectory is *OP-DP*. Then it is changed to *P1-DP* and the moving process is initiated again. Then again a collision is detected with the endpoint *P3* of a line segment and the trajectory is changed to *P3-DP*.

In the next step a new point is split from *OP* for the cases of adding a new point, making a new line segment or constrained edge. Split function is called with the trajectory *OP-DP* as parameters. This adds to the mesh one new point *MP* and two new zero-area triangles adjacent to the *SE* = *OP-MP* edge.

Then if required, the edge *SE* is converted to a constrained edge or a line segment. *SE* is

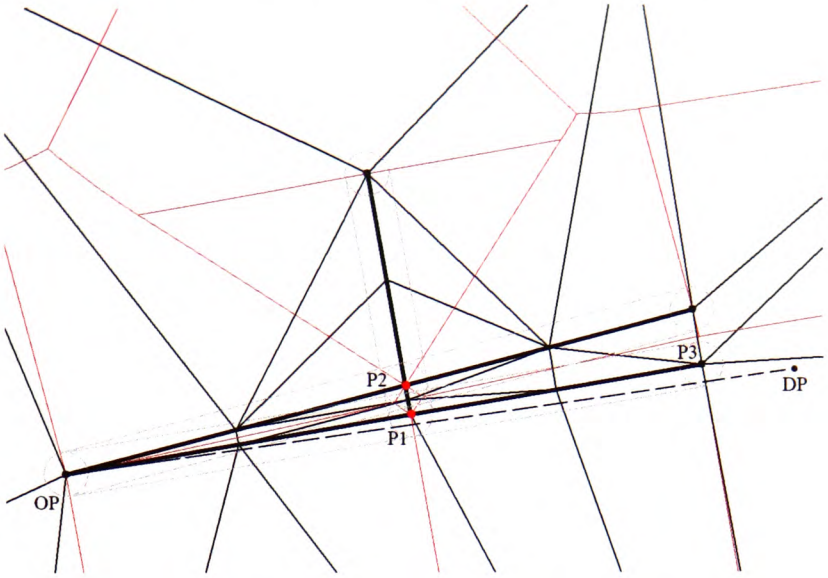


Figure B.2: A trajectory colliding with two endpoints of lines.

changed to constrained by setting its C flag to true. Alternatively the edge *SE* is converted to a line segment by adding two half lines joining *OP* and *MP* and six new "triangles", as described in Section 5.4.4. The new line segment will be identified using *HL* variable.

In the next step the real triangles located behind *MP* are identified by projecting their circumcentres (Voronoi vertices) onto the trajectory. Those will be ignored in the movement process, as the real triangles are used for obtaining new neighbours for the moving point and those have Voronoi vertices in front of the moving point. For each edge e originated at *MP* the real triangle is identified by three objects: $e.Dest = e.Sym.Oprev.Org$, $e.Sym.Oprev.Dest$ and $e.Sym.Oprev.Oprev.Dest$ and can be accessed using only one edge. The real triangles having the projections of their circumcircles behind the MP are stored in a list as one-edge entries for each stored triangle. Each edge e in the list corresponds to a triangle defined by $e.Org$, $e.Dest$, $e.Oprev.Dest$.

B.3 Detecting Topological Events

In the second main part of the process the potential topological events TE are tested to find the closest one and relocate the moving point to its position. The test trajectory is updated to *MP-DP* after each relocation of *MP*. The circumcircles of real and imaginary triangles are tested separately.

There are three possible locations of the circle related to the trajectory:

1. The circle does not cross the trajectory (no contact points)
2. The circle is tangent to the trajectory (one contact point)
3. The circle intersects the trajectory (one or two contact points)

In the third case, when one of the endpoints of the trajectory lies inside the circle there is only one intersection point. When both endpoints are outside the circle then the circle intersects the trajectory twice.

For the circles interacting with the trajectory the topological events are identified by testing positions of intersection points. A relative value of t is assigned for each intersection point CP. The value of t is:

- $t<0$ for projections behind MP
- $t=0$ for projections onto MP exactly
- $t\epsilon(0,1)$ for projections between MP and DP
- $t=1$ for projections onto DP exactly
- $t>1$ for projections behind DP

The circle intersecting the trajectory at the nearest position in front of MP (having the smallest t value) is selected as the one to be entered or left by MP. The constrained edges are ignored, as they cannot be swapped.

The collisions with points are detected and handled by modifying the trajectory. Collisions with line segments or constrained edges occur only when the trajectory of the moving point MP intersects them. Then the overlap of disks of MP and the colliding feature is accepted and all topological events are processed until in the CDT MP is connected to both endpoints of a constrained edge crossing the trajectory (Figure B.3a) or in the LSVD MP has the line segment crossing the trajectory as its only neighbour (Figure B.3b). Then no more more movement is possible towards the colliding segment so MP is moved back to its origin, the feature is split and the movement is attempted again to the split point and from there to the original destination.

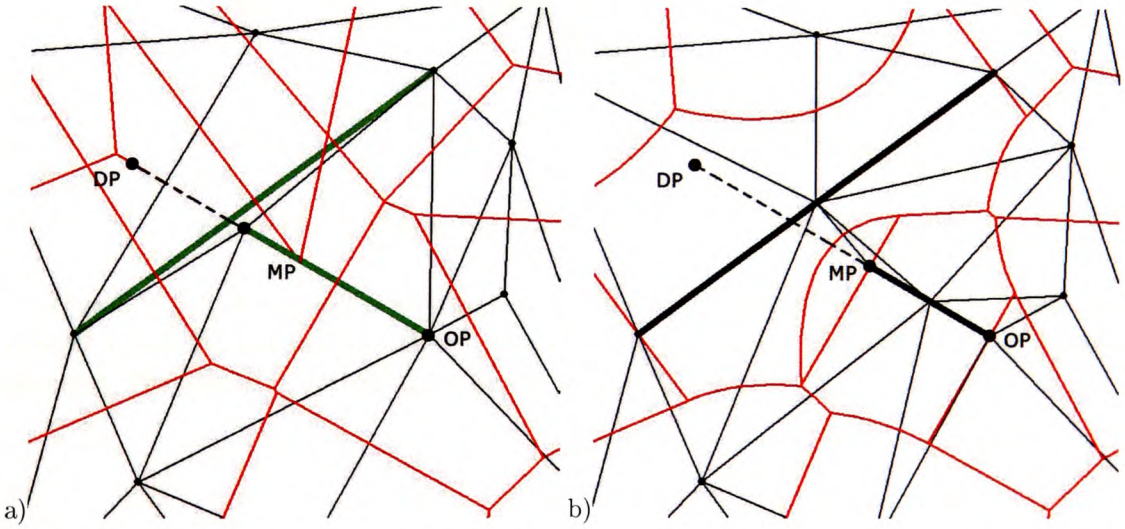


Figure B.3: Stopping configurations for collisions with linear features. a) Collision with a constrained edge. b) Collision with a line segment.

B.3.1 The Real Triangles

The real triangles are those adjacent to triangles surrounding MP. For each edge e originated at MP one real triangle RT is accessed. This triangle is determined by three objects: $e.Dest$, $e.Sym.Oprev.Oprev.Dest$ and $e.Sym.Oprev.Dest$, which can be points or line segments. The triangle RT is associated with a circumcircle having its centre CC at the Voronoi vertex of RT. If

any of the three objects is a line segment then the circle is tangent to that line segment. The edge *et=e.Sym.Oprev* is used for accessing that triangle and the circle can be described with three counterclockwise oriented points where the circle touches objects of the mesh: *v1=et.Org*, *v2=et.Oprev.Dest*, *v3=et.Dest*. If *et.Org* is a point then *v1=et.Org*. If *et.Org* is a line segment *LS* then *v1* is a point where the circumcircle touches *LS* and its coordinates can be obtained by projecting the circumcentre *CC* onto *LS*.

Firstly it is tested if *et.Org* is not a line segment crossing the trajectory (a collision with a line segment). If so and the position where they intersect is closer to *MP* than the nearest TE then this segment is saved. Also the position of the intersection is saved as the current TE. If currently there are no topological events detected closer to *MP* then that segment blocks the movement.

Later it is tested if the real triangle does not have *MP* as one of its vertices. When processing the real triangles, switching an edge connects *MP* to a new object and it is impossible to connect *MP* to itself. Figure B.4 shows a shaded real triangle *et.Org*, *et.Oprev.Dest*, *et.Dest* having *MP* as one of its vertices. When *et.Oprev.Dest=MP* it is not possible to switch *et* so such edge *et* ignored.

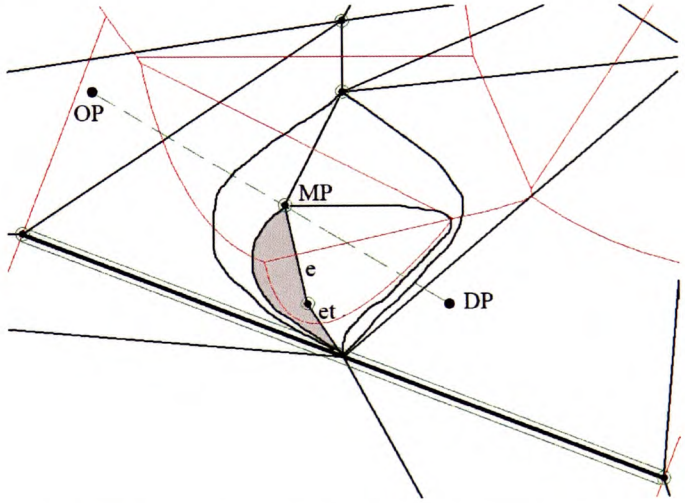


Figure B.4: A shaded real triangle has MP as one of its vertices.

Then it is tested if *et* is not an edge connecting two joined line segments, as in Figure B.5. In such a case the real triangle consists of two line segments *LS1* and *LS2* and their common endpoint. The edge *et* is ignored again, as it is not possible to switch it.

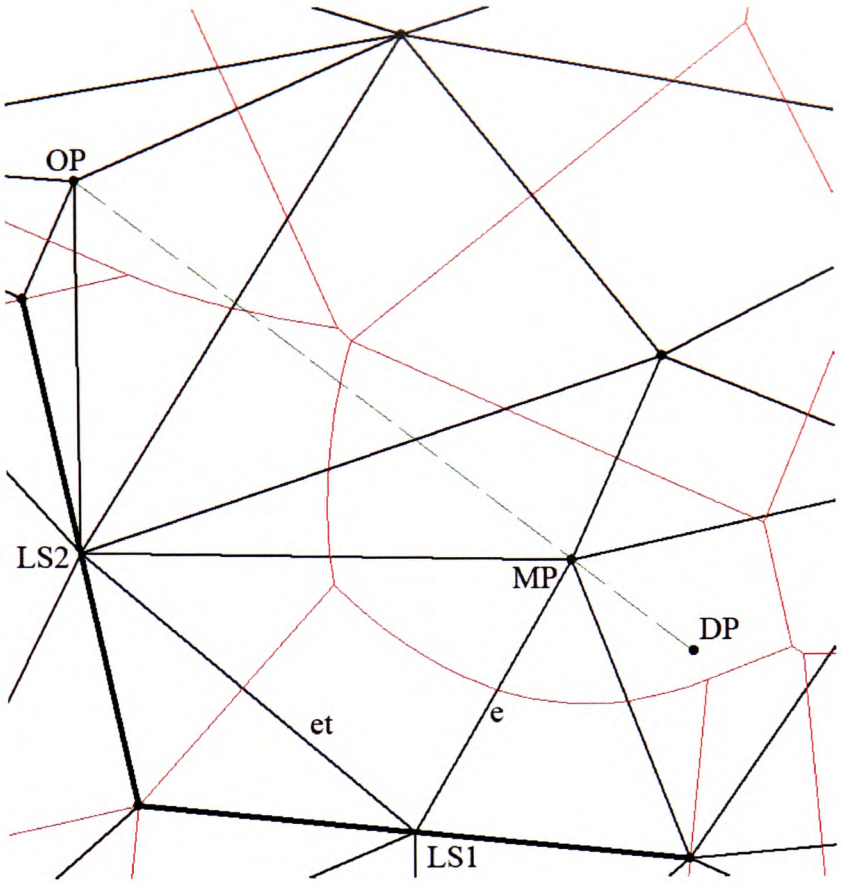


Figure B.5: Edge et connecting two joined line segments.

Edges "covering" line segments (connecting their endpoints with the interior) are also ignored, as they cannot be modified. Also the real triangles behind *MP* (stored in the *LE* list) are not processed, as only the triangles in front of *MP* can be used to obtain new neighbours.

Then if *et* is a constrained edge it is ignored, as constrained edges cannot be swapped. Additionally it is tested if constrained *et* does not intersect the trajectory, so the movement of *MP* to the other side of it is not possible without splitting it. If so the constrained edge *et* is saved and the intersection point position is calculated on *et*. It is compared with the current TE and saved as the current TE if it is the closest to *MP*.

After the initial tests processing of the circumcircle of RT is initiated. The circumcentre *CC* is available at *et.Rot.Org*. Due to the inaccuracy of the floating arithmetic the distance between *CC* and each of the nodes can be slightly different so the radius of *CC* is computed using the averaged distance from *CC* to each of the nodes of RT.

In the next step it is tested if *et* edge can be considered for the swap. This occurs if the circle contacts the trajectory - by touching or intersecting it. If a circle intersects the trajectory it can occurs in two places (or in one if one of the endpoints of the trajectory is inside that circle). The real circles are entered by *MP* so only the point of the first intersection is used as a place of a potential topological event. The relative value t of TE is computed for the contact location and compared with the current TE.

If t is found to be the nearest TE then additional validation tests are performed. Firstly the PAM test ("Point-Arc Matching") is performed. Figure B.6 shows an example of a situation where

the PAM test rejects an impossible swap. The real circle *v1*, *v2*, *v3* for the edge *et* intersects the trajectory *MP-DP* in two places, the nearer to *MP* is marked *CP* and it is the topological event being tested. Relocating *MP* to *CP* location and swapping *et* would join *MP* with *v3*, creating two incorrect new triangles with overlapping edges. If we look at the position of *CP* on the circle we notice that it is not between points *v3* and *v1*. This is the source of the problem as the circle can be only entered between points *v3* and *v1*, so *CP* can be only located on the arc joining those two points. This can be easily detected using the orientation test. *CP* is rejected when *v3*,*CP* and *v1* are clockwise oriented (when CCW(*v3*, *CP*, *v1*)>0). We can notice that when CCW<0 then *CP* and *v2* are on the opposite sides of *et* (*et* can be swapped), otherwise they are on the same side. The same test is valid when the objects are line segments using the points where the circle touches segments.

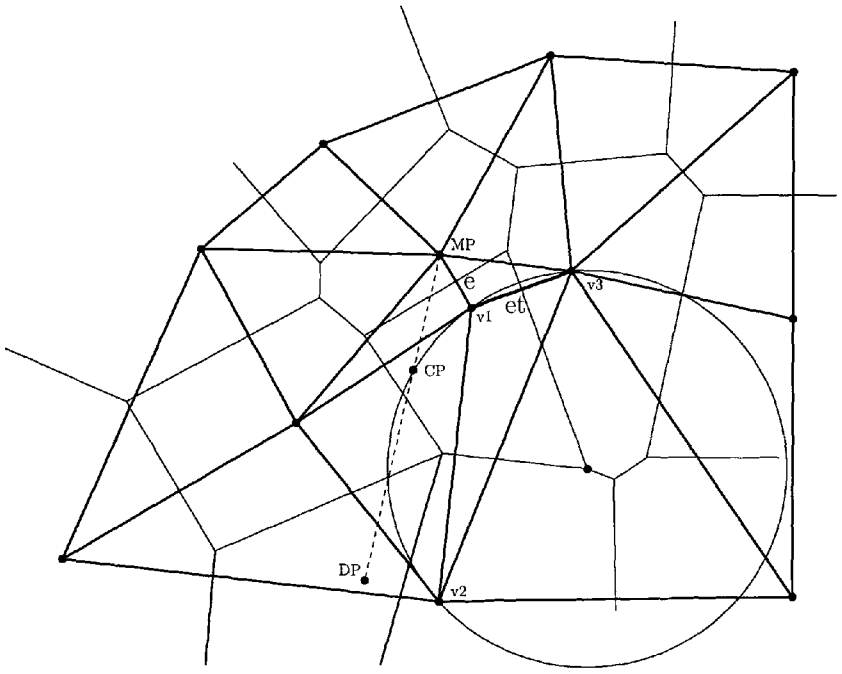


Figure B.6: The PAM test.

Finally, if the circle passes all validation tests the edge *et* is saved for the potential swap, the value t of the relative position of *CP* on the trajectory is remembered and the next real triangle can be tested.

B.3.2 The Imaginary Triangles

After testing all the real triangles the imaginary ones are processed. For each edge e originated at *MP* the imaginary triangle together with its circumcircle is defined by three objects: $o1 = e.Oprev.Dest$, $o2 = e.Dest$, $o3 = e.Onext.Dest$. Points where the circle is tangent to those objects, which can be points or line segment, are marked *v1*, *v2* and *v3* accordingly.

The circumcircle of an imaginary triangle, like the real circle, can cross, touch or avoid the trajectory. *MP* can only leave the imaginary circles so only positions of the second intersection with the trajectory (closer to *DP*) or where the circle touches the trajectory are considered as places of potential topological events.

For *MP* being a single point all imaginary triangles are tested. However, if *MP* is the head of an expanding line segment, as in Figure B.7, there are only two imaginary triangles that can become real (*o1,o2,o3* and *o6,o7,o8* in Figure B.7). There are no Voronoi vertices of *the MP's* cell behind the *MP*, they are in the position of *MP* (projection value *t=0*) or in front of it (*t>0*). Relocating *MP* means reducing the length of one of its Voronoi edges to zero. There are only two Voronoi edges *ve* that can be reduced and those are the first ones neighbouring the Voronoi cell of the line segment (Voronoi edges *ve2* and *ve7* in Figure B.7a). The circumcentres of those two circles lie on Voronoi edge of the edges joining two objects defining those circle, as the third object is the new line segment. Each circle is tangent to the trajectory and the position of the projection *CP* of the circumcentre is the location of a potential topological event and the new position of *MP*. If one of those two imaginary circles is identified as the one causing the closest topological event then *MP* loses a neighbour *e.Dest* after relocating it to *CP* and swapping edge *e* (edge *e7* in Figure B.7a). The object *e.Dest* (*o6* in Figure B.7a) becomes a neighbour of the new line segment (*o6* joined with the line by edge *e7* in Figure B.7b) and the imaginary triangle becomes a real one.

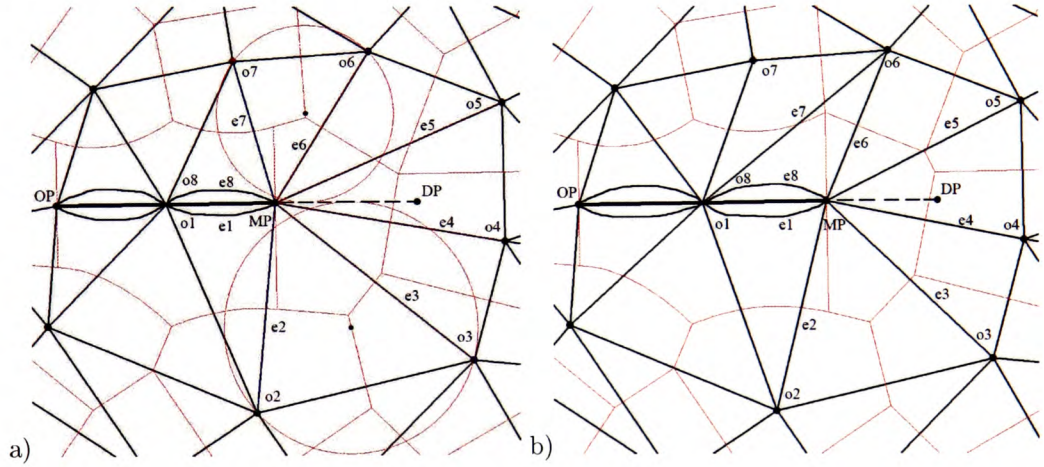


Figure B.7: Processing imaginary circles a line segment expansion process. a) Two imaginary circles that can be used in the iteration. b) After flipping edge e7.

The processing of the imaginary triangles starts with a collision test. If the object *o2* is a point located withing the trajectory range between *OP* and *DP* then a collision test is performed (collision tests with line segments were performed while testing the real triangles), measuring the distance from *o2* to the trajectory. If the distance is smaller than *2*DiskRadius* then the point collides with the trajectory - it is going to be hit by *MP* at some point. Then the relative position *t* on the trajectory is computed for the colliding point. If it is larger than the current TE value it is ignored, as there is a topological event before the collision. Otherwise the collision fact is saved, together with the colliding point and its relative position *t* along the trajectory, which becomes the current potential TE.

However there are cases when collisions are ignored, as changes of the trajectory might cause other collisions leading to infinite loops in the process. If a collision is detected with a point *P* when moving a point from *OP* to the *DP* location, the moving point *MP* is moved back and the trajectory is changed from *OP-DP* to *OP-P* and *P-DP*. However, it is possible that after relocating *MP* to the *P* location and attempting to move it towards *DP* a collision is detected with a point, which happens to be the previous origin point OP of the movement. Accepting this collision leads to another change of the trajectory to *P-OP* and *OP-DP*. After *MP* reaches *OP* the trajectory is

OP-DP, which is the initial trajectory and the process repeats. To avoid such cases after detecting a collision with a point *P* (Figure B.8a) it is tested if the potential trajectory *P-DP* does not collide with the origin point *OP*. If not the collision is accepted, otherwise the collision is ignored, as it would lead to an infinite loop. This leads to a temporary overlap of disks of points, as in Figure B.8b) which is fixed after *MP* reaches its destination (Figure B.8c).

When moving *MP* from *OP* to an existing point *DP* it is also possible that after detecting a collision with a point *P* (the configuration of nodes in Figure B.8c but with *MP* at the location of *OP*) and modifying the trajectory to *OP-P* and *P-DP* there is a collision detected with *DP* when moving from *OP* to *P*. This would change the trajectory back to *OP-DP* and lead to an infinite loop of collisions and trajectory changes. To avoid that after detecting a collision with a point *P* it is tested if moving *MP* towards *DP* will not cause a collision with the existing point *DP*, by computing the distance from *DP* to the potential trajectory *OP-P* and ignoring this collision when the distance is smaller than *2*DiskRadius*.

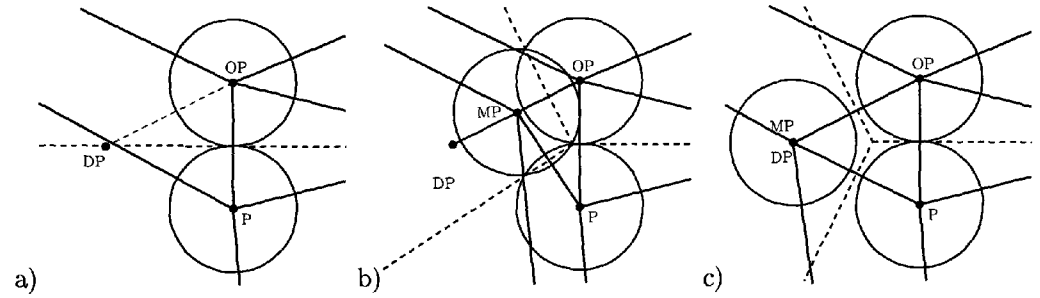


Figure B.8: An accepted collision case. a) The initial configuration. b) An intermediate configuration with MP colliding with P. c) The final configuration after MP reaching the location DP.

After the collision tests the circumcircle is calculated for the three objects. When expanding a new line segment the trajectory *OP-DP* is used as one of the objects, instead of the new line segment. This is because the imaginary circle is tangent to the trajectory, and the current line might be too short (or its length can be zero) to obtain a tangent circle. So in Figure B.7a instead of using objects *o1* and *o8*, a temporary segment *OP-DP* is used for calculating imaginary circles. The radius is computed averaging the distance from the obtained circumcentre *CC* to all three objects (or the trajectory).

Then the intersection of the circle with the trajectory is tested. Further processing is performed only when the circle crosses or touches the trajectory. The relative position of the contact *CP* is compared with the current nearest topological event. If it is nearer to *MP* then the PAM test is performed to test if the swap of the edge is possible. If PAM is successful then the topological event is accepted as the current nearest. The edge *e* is saved as the potential edge to be swapped, together with the new value of TE and the new potential position of *MP*. Then the rest of the edges is processed.

B.3.3 Processing Topological Events

After testing all real and imaginary triangles the results are processed. If there was a line segment or a constrained edge intersecting the trajectory detected while making a new line segment or constrained edge (*operation* 3 or 4) then it has to be split into two parts connected by a node, in order to allow *MP* to reach the *DP* destination through that new node. In such a case *MP* is

moved back to the origin position *OP* (which retracts the new line segment or constrained edge) calling MovePoint with the parameter *operation* set to 5 or 6 and *retractedObject* pointing to the retracted feature. This brings the diagram to the configuration before the feature expansion. Then splitting of the colliding feature *CF* is initiated. If the intersection point location is closer than *2*DiskRadius* to any of the endpoints of *CF* then *CF* will not be split and that endpoint is the point where *MP* is going to be directed. Otherwise *CF* is split at the intersection point location, which creates a new point *CLP* in the interior of *CF*. This is done by retracting one of the endpoints *CFE* of *CF* towards the intersection point location, by calling MovePoint with the parameter operation set to 6 (to detach it from its endpoint) and *retractedObject* pointing to *CF*. Then *CF* is reconstructed between *CLP* and *CFE*, calling MovePoint with *operation* set to 4 or 5 and *reconstruct* parameter set to true, so collision tests are not performed, as this feature had existed there already. After splitting *CF* into two parts, the process starts again and *MP* is moved again, this time from *OP* towards *CLP*. After merging *MP* and *CLP* the movement is continued from the location of *CLP* towards *DP*.

Figure B.9 presents a sequence of steps performed during a collision of two line segments. Construction of a line segment is attempted between a vertex *OP* and a *DP* location (Figure B.9a). The new line is expanded until no further movement is possible due to another segment with *LP1* and *LP2* endpoints crossing the trajectory at the *CLP* location (Figure B.9b). Then the new line is retracted back to the *OP* location (Figure B.9c) and the splitting of the line segment starts with disconnecting the segment from *LP1* endpoint and retracting it to *CLP* location (Figure B.9d). After that a new segment is created at the *CLP* location and expanded towards the *LP1* endpoint, which leads to splitting the *LP1-LP2* segment into two segments having a common *CLP* point (Figure B.9e). Then again a new segment is created at OP location and expanded towards *CLP* until no further movement is possible (Figure B.9f) and the new segment is merged with the *CLP* vertex (Figure B.9g). In the last step a new segment is extracted from *CLP* and expanded towards the initial destination *DP* (Figure B.9h).

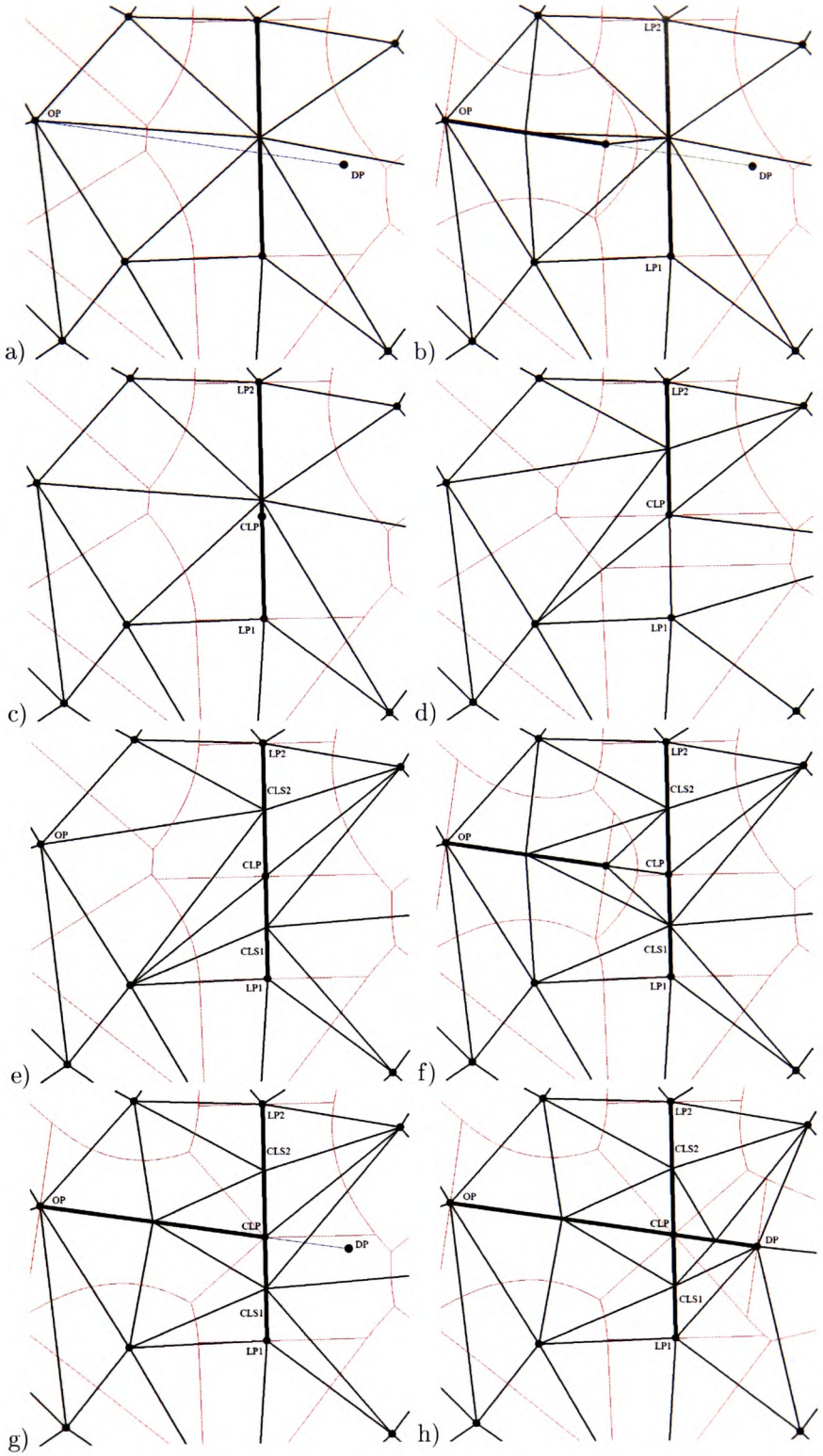


Figure B.9: A collision of a line segment with another line segment.

Otherwise, if there was a collision detected with a point *CLP* then *MP* is moved back to *OP* (additionally the list *LE* of the "behind" triangles is reset), and then the whole process is started again from *OP* with *CLP* as the destination. This leads to merging *MP* and *CLP*. After that the

process is continued from *CLP* towards the original destination *DP*. The result of such a situation for a line segment expansion is shown in Figure B.1.

If there was no collision found, but there was a topological event detected then *MP* is relocated to the position of the event. An appropriate edge is swapped, which in case of the real triangle leads to gaining a new neighbour, or in case of an imaginary triangle to losing one. Then the Voronoi diagram and the list of the real triangles behind *MP* is updated.

Alternatively, if there were not any topological events detected between the *MP* and the location of *DP* then *MP* is relocated to *DP*, just by changing its coordinates, and the Voronoi diagram is updated. If there was a node P already at the location *DP* then *MP* and P are merged together.